



Calhoun: The NPS Institutional Archive
DSpace Repository

Theses and Dissertations

1. Thesis and Dissertation Collection, all items

1980-03

Petri-Net simulations of communications networks

Jennings, Stephen Craig; Hartel, Robert John

Monterey, California. Naval Postgraduate School

<http://hdl.handle.net/10945/19009>

This publication is a work of the U.S. Government as defined in Title 17, United States Code, Section 101. Copyright protection is not available for this work in the United States.

Downloaded from NPS Archive: Calhoun



<http://www.nps.edu/library>

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

PETRI-NET SIMULATIONS
OF
COMMUNICATIONS NETWORKS

by

Stephen Craig Jennings

Robert John Hartel

March 1980

Thesis Advisor:

Lyle A. Cox

Approved for public release; distribution unlimited

T195855

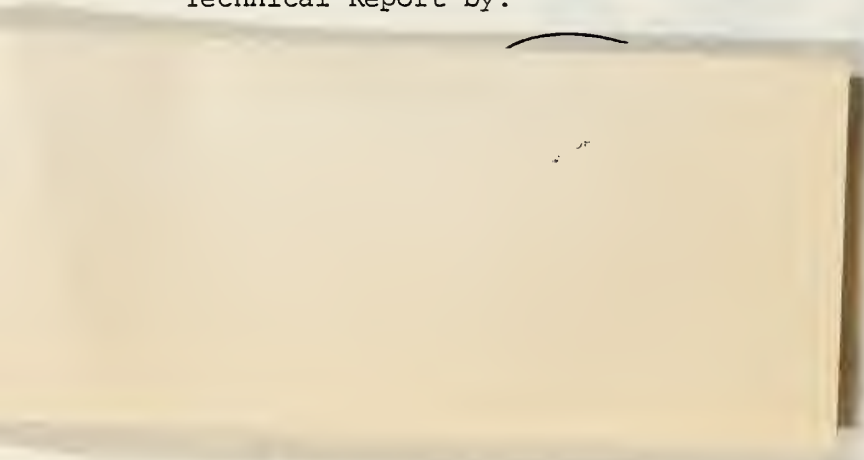
NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral J. J. Ekelund
Superintendent

Jack R. Borsting
Provost

Reproduction of all or part of this report is
authorized.

Released as a
Technical Report by:



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NPS52-80-003	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Petri-Net Simulations of Communications Networks		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; March 1980
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Stephen Craig Jennings Robert John Hartel		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93940		12. REPORT DATE March 1980
		13. NUMBER OF PAGES 178
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Simulation Petri-Net Evaluation		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) Current technologies in the fields of telecommunications and computer processing are becoming increasingly integrated to the extent that "distributed" computer networks are assuming key roles in communications. The complex computerized systems necessary to support modern military command and control requirements are expensive. Designing such systems by trial and error is not feasible, yet no other viable alternatives exist. This thesis offers an original methodology for evaluating the predicted		

performance of military automated systems. Using Petri-Nets as a modeling tool, computer simulations with color graphics output are performed to demonstrate the feasibility of this approach as a systems design tool.

Approved for public release; distribution unlimited.

PETRI-NET SIMULATIONS OF COMMUNICATIONS NETWORKS

by
Stephen C. Jennings
Captain, United States Marine Corps
B.S., United States Naval Academy, 1971
and
Robert J. Hartel
Captain, United States Army
B.S., Texas A&M University, 1972

Submitted in partial fulfillment of the
requirements for the degrees of

MASTER OF SCIENCE IN SYSTEMS TECHNOLOGY (C3)
(Stephen C. Jennings)
MASTER OF SCIENCE IN COMPUTER SCIENCE
(Robert J. Hartel)

from the
NAVAL POSTGRADUATE SCHOOL
March, 1980

ABSTRACT

Current technologies in the fields of telecommunications and computer processing are becoming increasingly integrated to the extent that "distributed" computer networks are assuming key roles in communications. The complex computerized systems necessary to support modern military command and control requirements are expensive. Designing such systems by trial and error is not feasible, yet no other viable alternatives exist. This thesis offers an original methodology for evaluating the predicted performance of military automated systems. Using Petri-Nets as a modeling tool, computer simulations with color graphics output are performed to demonstrate the feasibility of this approach as a systems design tool.

TABLE OF CONTENTS

I.	INTRODUCTION.....	9
II.	NETWORKS: MOTIVATION, TAXONOMY AND PERFORMANCE.....	14
A.	INTRODUCTION.....	14
B.	MOTIVATION FOR THE ANALYSIS OF PERFORMANCE.....	14
C.	TAXONOMY.....	16
1.	Networks.....	16
2.	Computer Communication Networks.....	16
3.	Further Reading.....	18
D.	PERFORMANCE PREDICTION AND MODELING.....	18
E.	SUMMARY.....	21
III.	MILITARY APPLICATIONS OF DISTRIBUTED SYSTEM TECHNOLOGY.....	22
A.	INTRODUCTION.....	22
B.	PACKET SWITCHING.....	22
C.	THE ARPANET.....	25
D.	MILITARY APPLICATIONS.....	27
E.	THE ALOHA SYSTEM.....	28
F.	PACKET RADIO INTRODUCTION.....	29
G.	CHANNEL ACCESS SCHEMES.....	31
1.	Category I.....	32
2.	Category II.....	36
3.	Category III.....	37
H.	CURRENT PROGRAMS.....	38
1.	PLRS/JTIDS Hybrid.....	39

2.	The San Francisco Bay Experimental	
	Packet Radio Network.....	39
3.	The Fort Bragg Testbed.....	41
I.	FUTURE IMPLICATIONS.....	43
1.	Chain of Command.....	44
2.	Network Management.....	45
3.	How Much Redundancy?.....	46
4.	Propagation Loss Due to Higher Frequency.....	46
5.	Management Information Systems.....	47
6.	Interoperability.....	47
7.	Voice vs. Data Circuits.....	48
8.	System Cost.....	48
IV.	AN INTRODUCTION TO PETRI-NETS.....	49
A.	INTRODUCTION.....	49
B.	HISTORY.....	49
C.	HOW PETRI-NETS WORK.....	51
D.	A SIMPLE EXAMPLE.....	54
E.	ADVANTAGES AND DISADVANTAGES.....	54
V.	PETRI-NET SIMULATION OF COMPUTED COMMUNICATION	
	NETWORKS.....	60
A.	INTRODUCTION.....	60
B.	TYPES OF NODES.....	61
C.	A SIMPLE APPLICATION EXAMPLE.....	62
D.	RANDOMNESS IN PETRI-NETS.....	64
E.	MEMORY STORAGE REPRESENTATION.....	67
F.	SYSTEM LOAD AVERAGE.....	67
G.	TIME REPRESENTATION.....	68

VI. THE EXPERIMENT.....	71
A. DESCRIPTION.....	71
B. RESULTS.....	73
VII. RECOMMENDATIONS AND CONCLUSIONS.....	77
A. RECOMMENDATIONS.....	77
B. CONCLUSIONS.....	80
APPENDIX A - USER INSTRUCTIONS FOR PETRI-NET SOFTWARE.....	82
A. INTRODUCTION.....	82
B. THE INPUT FILE.....	82
1. Places.....	84
2. Transitions.....	85
3. Initial Marking.....	86
4. Execution.....	87
C. THE OUTPUT FILES.....	87
D. USER OPTIONS.....	88
1. Choice of Programs.....	88
2. User Questions and Responses.....	88
3. The Highlighting Feature.....	91
E. UNIQUE INSTRUCTIONS FOR VERSION 3.....	91
APPENDIX B - RUN03.....	97
APPENDIX C - RUN03A.....	98
APPENDIX D - RUN03B.....	99
APPENDIX E - RUN03C.....	100
APPENDIX F - RUN20.....	102
APPENDIX G - THESIS FILES.....	108
COMPUTER PROGRAM - "simulator".....	109
COMPUTER PROGRAM - "transgraph".....	119

COMPUTER PROGRAM - "linkgraph".....	147
LIST OF REFERENCES.....	174
INITIAL DISTRIBUTION LIST.....	176

I. INTRODUCTION

As military planners look forward to the design of future command, control, and communications (C3) systems, several key factors should be at the forefront of their thinking. First of all, it is readily apparent that there has been a proliferation of computer resources for military applications. Even the most "tactical" of systems today is becoming a sizeable collection of computers, databases, sensors and information-handling equipment.

Secondly, one can sense that the fields of telecommunications and computer science are becoming increasingly integrated. Although these once-separate disciplines have very different histories and traditions, they are experiencing a technological convergence which is having far-reaching implications for both the military and civilian societies. Today the concepts and techniques of computer processing have been integrated with communications to the extent that both fields share the same kind of logic, storage, switching and transmission. Because information handling systems now employ telecommunications and information in such an intimate mixture, it is difficult to distinguish what in the system is computer processing and what is communications.

A third key factor should be a realization that certain concepts of computer communication networks and distributed

system architectures offer advantages for military applications due to the potential for survivability and lack of centralization.

Computer networks are often created spontaneously by combining computers and communications. The growth of computer networks is one of the significant outcomes of the convergence of the two disciplines. An extensive array of computing resources can be connected over a wide geographic area via telecommunications channels. The potential architectures for such networks are limitless when one considers the variety of hardware, software, protocols and geographic distributions that might comprise the system.

Another key factor is that system planners need to focus their efforts upon total system integration. Tactical automated systems tend to be developed individually to meet unique mission requirements. For instance, separate systems are typically justified and developed for missions of fire control, air defense, intelligence, personnel management and logistics, etc. While these separate systems may perform satisfactorily alone, there is difficulty in providing a suitable management information system by which the overall commander or decision-maker can have access to the information in all these separate systems in a format that is easy to understand and use.

Another important consideration is that the proliferation of automated systems places a severe burden upon the communications equipment that links systems

together. There is a tendency for automated systems to be separately developed with insufficient emphasis placed upon the communications equipment that will transmit the information. In other words, the sensors and processors of a system may work splendidly, but planners must not take it for granted that the data will arrive at the correct destination in an error-free condition.

The communications equipment and channelization that carry the information must be as carefully engineered as the other components of the system. In addition, the data on the channel must be in a format that is compatible with other systems. The U.S. Army is coming to grips with the fact that if the some 50 tactical automated systems on the drawing boards were to be fielded for use at the corps level, there exists no communications equipment capable of carrying the vast volume of information these systems would generate. In addition, when the communications equipment must operate with specifications of transmission security, jam resistance, and low probability of intercept in a severe electronic warfare threat environment, the design difficulties are considerably magnified.

Lastly, military planners should be concerned about being able to accurately predict system performance. An accurate predictor of system performance is needed for use by those in procurement duties to ensure that performance specifications given by contractors will in fact prove to be true when a new system is fielded. Managers of currently

operational systems also require this service. They are asking in the course of daily duties such questions as: What would be the effect of increased buffer space at this busy location? What would be the impact on total system performance if a particular node or link in the network is removed? Such a desire for prediction of system performance has created great emphasis upon modeling techniques and computer simulations of systems to answer these questions.

All of the above factors have given impetus to this thesis. A particular modeling tool which addresses these needs is described (the Petri-Net) and is implemented to facilitate communications network modeling.

The paper presents three primary points of original work:

1. It is demonstrated that automated networks can be meaningfully modeled with the use of Petri-Nets.

2. It is shown that Petri-Net models of networks can be adapted for effective computer execution and display on a color graphics terminal. Simulations which incorporate graphics output are more easily understood and have considerable educational value.

3. The results of this research indicate that the implementation of such a modeling technique in a production environment as a predictor of system performance is feasible.

The above 3 points, although successfully implemented in the Naval Postgraduate School C3 laboratory, represent a

new area of research in a preliminary stage of development. Future investigation is required to expand and validate this approach.

II. NETWORKS: MOTIVATION, TAXONOMY AND PERFORMANCE

A. INTRODUCTION

Predicting the performance of an automated network can indicate a measure of a system's effectiveness and efficiency.[3] Evaluating a system's performance is a complex task. This task often requires modeling. Many performance modifications are more suitably performed on models than the actual system, because "trial and error" production is not feasible economically. This chapter discusses the motivations for network design and the practicalities of predicting network performance.

B. MOTIVATION FOR THE ANALYSIS OF PERFORMANCE

The magnitude of information processing in the United States is unprecedented and still growing. Computer processing and communications make a major portion of currently accessible information available to federal agencies and commercial businesses. As the country is becoming increasingly dependent on the need for information, the existence of reliable, effective computer communication networks is essential to transport computer-processed information.

With the added cost of energy, the attractiveness of moving resources to the user via computer communications networks is apparent.[5] The performance evaluation of these networks is, therefore, a measure of the system's ability to

meet user requirements.

Adequate performance evaluation tools currently do not exist. Therefore, decision-makers responsible for selecting new systems usually rely on the designer's claims as to performance, and procure systems accordingly.

Both the civilian and military communities place very heavy demands on communication's facilities during crisis situations. Throughout history existing systems have often not been sufficient to carry such communications traffic.

[8]

The civilian community shares these problems. Many networks are engineered to carry mean traffic loads and are not planned for crisis contingencies. The commercial telephone network, for instance, is inundated with traffic Christmas and Mother's Day, severely degrading system performance. The disruptive affects of a national emergency on all types of automated networks can only be imagined.

In the Defense Communications Agency (DCA), work is presently under way to develop tools for evaluating the performance of planned computer communication networks. This effort is underway because development of these expensive systems by "trial and error" can no longer be afforded. The necessity to have confidence in the system's ability to meet design specifications before production is essential, since normal federal procurement cycles stretch out over eight to ten years. This confidence can be insured by utilizing effective performance evaluation tools.

C. TAXONOMY

A great deal of ambiguity exists in the jargon of the networking field. [7] Therefore, several recurring network descriptions are defined in this chapter to provide a consistent vocabulary. Using these definitions, network issues can then be succinctly conveyed to the reader.

1. Networks

The term "network" conveys the concept of individualized cells and a degree of interconnectivity. A network exists for the purpose of achieving a desired objective. The individual characteristics of networks are related to network topology, hardware configurations, and software control features designed to accommodate the user's requirements.

2. Computer Communication Networks

A "computer communication network" is a system consisting of one or more computers and terminals, and a communications subsystem which connects them. [7] The primary purpose of this network is to facilitate the efficient flow of data, and provide the required supportive processing functions. The communications subsystem consists of transmission facilities and associated communications processors. Communications processors are computers dedicated to exclusively handling communications tasks.

The classification of computer communication networks often centers around the network topology, network connectivity, switching protocol and the degree of

implementation of system-wide control features.

The term "centralized computer communication network" is used to define a network that possesses a high degree of centralized functions. Another network classification, "distributed systems", is used to describe a low degree of centralized functions.

The differences between centralized computer communication networks and distributed systems reveal themselves in the degree to which system functions are distributed. There are no concise metrics which delineate the exact classification of a network. The interpretation is subjective.

Further elaboration on specific advantages and disadvantages of the distribution of system functions and the definition of distributable functions can be found in CYSFER [6].

Centralized computer networks essentially formalize the system control structure into pre-selected system control nodes. These nodes contain what is often referred to as the network control programs (NCP). The NCP can control the processing, database management and communications management functions. This type of networking characteristically has a low degree of fault tolerance because network direction emanates from only one node imposing network restrictions should this node be damaged.

Distributed systems, however, are typified by the distribution of system management functions. Although no

"pure" distributed systems exist in reality, the term is widely used to indicate a high degree of distributed functions within a network. This "peer" structure acts in a cooperative sense. Routing algorithms, for example, rely on information that is "cooperatively" passed from node to neighboring node, thereby deriving information, not on a global basis but on a localized one. Distributed systems have a high degree of survivability and are, therefore, more fault tolerant than centralized computer communication networks. The advantage of distributed systems is, however, partially offset by the increased "overhead" necessary to coordinate system functions.

3. Further Reading

Numerous terms used in networking contribute to confusion due to the lack of an industry-wide standardized taxonomy. The intent here was to define certain keywords used throughout this thesis. Further clarification of system's taxonomy, although important, is not discussed. Taxonomical studies recommended are [2], [9] and [23].

D. PERFORMANCE PREDICTION AND MODELING

"Performance" is defined as the degree to which the system fulfills user requirements. In terms of networks, these measures are often determined by the network's workload capacity or throughput in the sense that the network can first perform the desired functions and secondly perform with a degree of timeliness. [4]

The ideal means of measuring network performance is to extract data from the system itself. The collection of data from existing networks is often difficult. The performance testing of networks under heavy traffic loads or those operating in a degraded mode is a sensitive matter, because interference of the monitoring equipment can not be tolerated. Networks in design phases, of course, can not be measured and require alternative assessment methods.

The modeling of existing and planned networks has become an important component of performance evaluation. As an added benefit, modeling also facilitates the understanding of a system's design and interrelationships.

The modeling process can follow one of several different techniques or a combination thereof. The primary techniques are: 1) mathematical modeling and 2) simulation.

Mathematical modeling employs theories of queuing and flow by describing certain network characteristics in sets of equations. The process is, however, complex and often assumes away critical parameters. The primary disadvantage with mathematical modeling appears to be just this problem of assumptions. Too many assumptions impose an unacceptable degree of abstraction. Although the validity of mathematical modeling techniques has been confirmed [4], the methodology is often understandable only by the modeler.

Simulation, the second alternative, leads the modeler to numerous techniques. A simulation is an abstraction of concepts pertinent to the problem being studied, and upon

which the modeler can apply varying experimental variables. The model simulates only those features the modeler feels are relevant to the problem. Herein lies the critical danger of simulation. The danger may best be expressed by the question, Does the model bear relevancy to the real problem? [10]

The major problem of simulation, as well as analytical models, is therefore the validation of the model. Many simulation experts talk in terms of performance reliability factors but fail to state that these factors may be just derived from outputs of a model, and the closeness of the model to reality may or may not be substantiated.

The modeling process itself consists of the construction of the model, followed by the validation of the model, and finally modifications to the model based on results of the validation process.

Once the model has been defined, simulations are run with the model to evaluate network behavior. The problem is to predict performance of real networks by evaluating behavior on the network models. The simulations then provide a means by which network design deficiencies can be identified and corrected.

Another issue in simulation analysis is the area of overdesign. Simulations should indicate those systems components that do not add to the capabilities of the network. This analysis can measure the device utilization of specific components within the network.

The key measurement of a computer communication network is the network's workload capacity or throughput. Throughput is generally measured in number of message units per time period, and provides a measure of effectiveness and efficiency of the system. The parameters involved in throughput are: 1) network configuration (topology), 2) network control algorithms, and 3) network reliability.

E. SUMMARY

A solution to predicting performance in systems where the collection of data is difficult or the system is in design stages, is to build a model of the system. The model could then be tested over the entire spectrum of performance specifications.

The goal of performance evaluation is the prediction of the degree to which the system fulfills the intended objectives. The major concern in computer communication networks is the degree to which the network can perform the task and the degree of efficiency with which the task is completed. A by-product of the evaluation should be the identification of areas of over and under-design. Once design failures have been identified, design tradeoff decisions can then be made. This process of performance analysis is aimed at optimizing the existing or planned network's performance and ensuring that the performance meets the contracted user requirements.

III. MILITARY APPLICATIONS OF DISTRIBUTED SYSTEM TECHNOLOGY

A. INTRODUCTION

This chapter discusses current military programs and research efforts that are applying the concept of automated networks to tactical missions. It is written to give the reader some background information on programs and terminology. With this background, the applicability of the simulations described in later chapters will be clearer.

B. PACKET SWITCHING

The transmission of computer to computer digital messages has had significant impact on communications switching techniques. In fact, the concept of packet switching was invented to a large extent because of the unique requirements of computer based systems. Packet switching was designed as an alternative to circuit switching.

The circuit switching technique of older communications systems is a method of establishing a route for communications traffic whereby a complete link between the calling and receiving station is set up and maintained exclusively for the exchange of those two stations. The connection is maintained until one of the stations breaks off transmission or reception. A technique such as this tends to be wasteful in computer communications because computer communications are typically "bursty" in nature;

that is, the messages are very short in duration and require fast responses.

Packet switching is designed to make efficient use of a communications channel when the traffic is bursty. [11] In this technique messages are divided into discrete "packets." A packet is a block of information containing a fixed number of bits. Each packet contains the text of the message plus a control header. The header contains enough information (for example, source, destination, routing plan, message sequence number, etc.) to guarantee the packet will arrive at the proper destination. In addition, there will usually be some checks on each such block, so that any switch through which the packet passes may exercise some degree of error control.

Figure 1. shows a typical composition of a packet of information bits. This particular example is taken from a packet radio network. [16]

In a packet-switching network, the packet represents the fundamental unit of transportation. One message may be broken into several packets and each packet may be independently routed to its final destination. Of course, at the destination the packets must be re-assembled in the correct order to reconstruct the original message.

Because packets of the same message can be sent by different routes, congestion on the network can be decreased. Each packet contains its own control information in the header, and there are no lengthy connection and disconnect times as in the case of circuit switched systems.

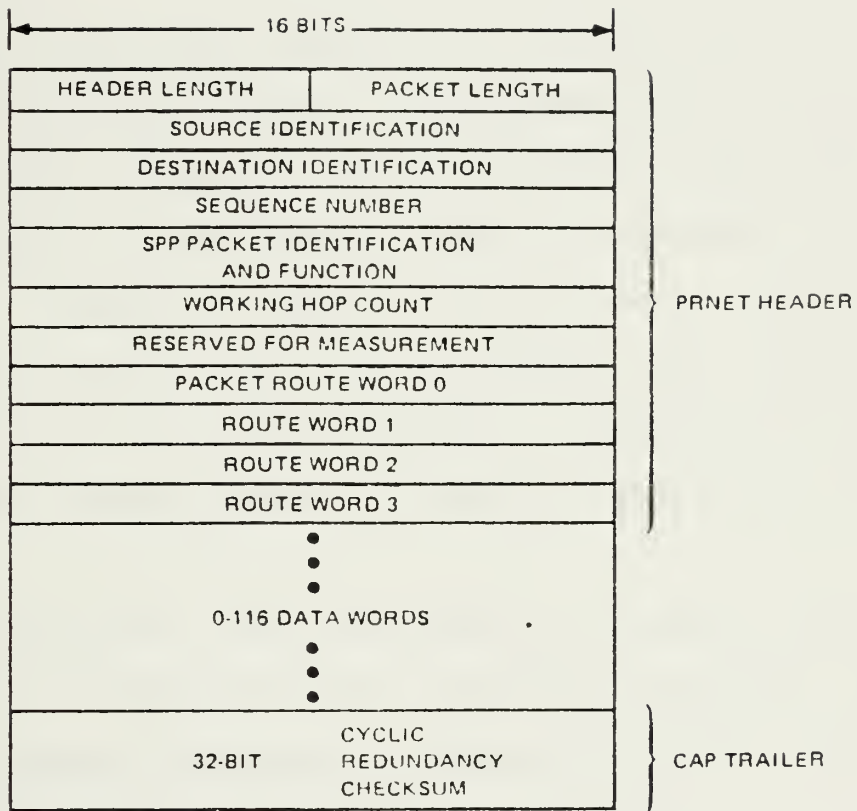


FIGURE 1 FORMAT OF PRNET PACKET (PRCAP3)

It should be noted that messages which are broken into packets are only meaningful within the network. When packets are passed through gateways into other packet-switched networks, a new intra-net level of protocol is required.

C. THE ARPANET

Perhaps the best known example in the military of a large scale distributed system is the ARPANET. This research effort has been sponsored by the Defense Department's Advanced Research Projects Agency (ARPA). The ARPANET is a non-secure, packet-switched, distributed computer communication network which links together the computing facilities at universities and military installations across the continental United States and reaches overseas to London and Hawaii.

The justification and advantages of a network such as the ARPANET are summarized in the following excerpts from the text Computer-Communication Networks written in 1973:
[1]

"One of the most successful aspects of the experiments in the use of time-shared computer systems conducted during the past decade was the ability to share computing resources among all the users of the system. Controlled sharing of data and software, as well as the sharing of the time-sharing system hardware, has led to much higher programming productivity and better overall utilization of the computing and user resources."

Even in these time-sharing systems, however, the system capacity was simply not large enough to perform all the storage requirements and computing potential that a decision maker required. There was the lack of a large enough community (critical mass phenomenon) in a single application area. Although it is possible to physically transfer programs or data from one community to another, this causes restrictions in language standards and hardware systems.

To quote further: "A viable alternative to program transferability, while permitting full resource sharing, is to provide a communications system that will permit users to access remote programs or data as if they were local users to that system. In addition, it should be possible for a user to create a program on his local machine that could make use of existing programs in the network as if they were available on his local machine. Rather than trying to move the programs from machine to machine, the network would allow the user or his program to communicate with a machine on which the program already executes. If enough machines can be connected into such a network, the total community in any particular application area would be sufficiently large enough to reach critical mass." [1]

This, essentially, is the rationale for military applications of the ARPANET research, both on the strategic and tactical level.

D. MILITARY APPLICATIONS

We have spoken in a previous chapter of the tremendous growth of automated data systems. Tactical data systems, be they for the purpose of intelligence, personnel management, fire direction, logistics, or command and control, tend to be engineered and developed separately without the consideration of total military mission. Nevertheless, the battlefield commander cannot make wise decisions based on input from just one of these systems. There is a good deal of interplay between all of these functional areas, and all must be considered.

One single tactical computer in a single command post could not be built to store and update all of the data represented in the combined systems. And if it could, such centralization would be unwise. The answer seems to be a distributed computer network built to interconnect and share the resources of the individual systems. This means overlaying an ARPANET-like architecture onto a series of distributed processors on the battlefield.

The ARPANET distributed architecture geographically separates data bases and computing resources. This distribution tends to decentralize a network, moving from a traditional hierarchical configuration to a grid or mesh-type configuration. Such decentralization is important to overall system survivability and reliability. A network architecture such as this is capable of remaining operational if one or more nodes is rendered

non-operational. The architecture, combined with packet switching technology, allows for sufficient alternate routing capability to ensure a robust system. It offers significant improvement over some of the present "backbone" (hierarchical) systems in which the failure of one node along the chain would completely disrupt communications on the entire network.

It is obvious that a mobile and tactical application of ARPANET technology would offer new challenges to system engineering. Most apparent is the fact that ARPANET sites are interconnected by high speed, low-error, fixed telephone circuits. This kind of interconnectivity is not possible on a dynamic battlefield. The only other alternative is to utilize mobile, digital radio equipment to achieve connectivity.

E. THE ALOHA SYSTEM

Several years ago researchers at the University of Hawaii began work on such hardware. Because there was an unusually high error rate on the local telephone lines, remote users of the university computer were unable to effectively communicate with the computing facility. This led to a research program to investigate the use of burst radio transmission in place of telephone lines for error-free, line-of-sight communications to the computer center. The resulting effort became known as the Aloha System, a series of packet-switched, ultra-high frequency (UHF), radio terminals. [13]

The Aloha System is essentially a broadcast, multi-access network. The "broadcast capability" of a radio channel implies that a signal generated by a radio transmitter may be received over a wide area by any number of receivers. "Multi-access capability" of a radio channel means that any number of users may transmit over a common channel. Hence, all users within line-of-sight of one another form a network that is completely connected, independent of the number of users.

F. PACKET RADIO INTRODUCTION

Research done at the University of Hawaii led to the development of packet radio. Packet Radio extends the Aloha System to military uses. The goals of military experimental versions of a packet radio system differ from those of the original Aloha net in the following areas: [13]

(1) Distributed control of the network management functions should be provided among multiple stations for reliability, and the use of a netted array of possibly redundant repeaters for area coverage as well as for reliability should be included.

(2) The system should use spread-spectrum signaling for coexistence with other possibly different systems in the same band and for anti-jam protection. Surface acoustic wave technology has become a viable current choice for matched filtering in the receiver.

(3) The provision of authentication and anti-deception mechanisms is required.

(4) System protocols should be incorporated that perform network mapping to locate and label repeaters, route determination and resource allocation, remote debugging, and other distributed network functions.

(5) The use of various implementation techniques to provide efficient operational equipment such as repeater power shutdown except while processing packets should be included.

Because packet radios operate within a broadcast network, and all the network radios use a common frequency band (1710-1850 MHz), this technology has some favorable implications for frequency management and frequency conservation. According to current military doctrine, the frequency spectrum is allocated roughly in accordance with each user's stated requirement. In an Army or Marine Division this results in a frequency management problem of too many nets requiring too few frequencies and a constant threat of degraded communications due to mutual interference problems. Once a frequency is allocated for a particular mission, it is not available for use by others in the same area.

This might be an effective management technique if each assigned band were actually used most of the time. In practice this is not usually the case, and much of the frequency spectrum is idle (not engaged in carrying traffic).

A broadcast network in which a number of users share a common broad frequency band offers improvement to this situation. The limited frequency spectrum could be used more efficiently if (1) the shared frequency band was wide enough to allow all users to transmit required traffic, (2) a channel access scheme was defined such that all users could access the channel when needed while at the same time allowing little or no mutual interference, and (3) the channel usage was high enough to ensure minimum empty time when the channel was not in use.

G. CHANNEL ACCESS SCHEMES

One primary means of categorizing radio broadcast systems is the method employed for channel access. As mentioned earlier, packet communications have found important applications in ground-based radio information distribution, and in this situation there exists a common broadcast channel that is available and shared by a multiplicity of users. Because these users demand access to the channel at unpredictable times, some access scheme must be introduced to coordinate their use of the channel in a way which prevents degradations and mutual interference.

A large number of channel access ideas have been invented, analyzed and described in current literature. For a summary of these schemes, see [14]. All of these schemes, however, might be placed in one of three broad categories. [15] Each category has its own advantages and costs.

1. Category I

The first category involves random access contention schemes whereby little or no control is exerted on the users in accessing the channel. This results in the occasional collision of packets on the air. A collision implies that at least one colliding packet is unintelligible and that channel usability for the time of the collision may be lost. Access schemes which fall into this first category are the pure Aloha, the slotted Aloha, and to a lesser extent, Carrier Sense Multiple Access. These are the access methods used by packet radio systems.

To better understand random access contention schemes, consider the example shown in Figure 2. There are some number of users, each of which transmits some number of packets of time duration " τ " at random times. Line four, labeled SUM, indicates the total traffic that all the users attempt to send in a fixed time.

In this example all traffic is able to be transmitted without conflict except for the collision indicated by the hashed area. These two packets (P-1 and K-1) may both be unrecognizable to the receiving station (at least one will be unreadable) and both could require retransmission.

In the non-slotted Aloha random access technique, packets are transmitted as soon as they reach the top of the transmit queue at the radio. No consideration is made of current channel activity. Therefore, they risk collision

with other packets on the air.

In the slotted Aloha method, time is broken into discrete quantities equal to the maximum time of propagation within the network. Users are restricted to transmission only at the beginning of each time slot. Again, collisions may be frequent.

In the carrier-sense mode, the radios listen before they talk and thereby reduce the risk of packet collisions. The radio senses the state of the channel before transmitting. If the channel is occupied, the radio waits a random amount of time and senses the state of the channel again before attempting to transmit.

An important measure of performance for evaluating these random access techniques is "throughput." Throughput in packet radio technology is defined slightly differently than the definition given in Chapter II. Here it is defined as the percentage of time that the channel is actually occupied by useful traffic. Or, to put it another way, it is the message density on the channel. Is the channel carrying useful traffic (non-colliding packets) most of the time, or is there a lot of time wasted between packets when the channel is empty?

Figures 3 and 4 show how throughput is calculated in both techniques.

RANDOM ACCESS SCHEMES

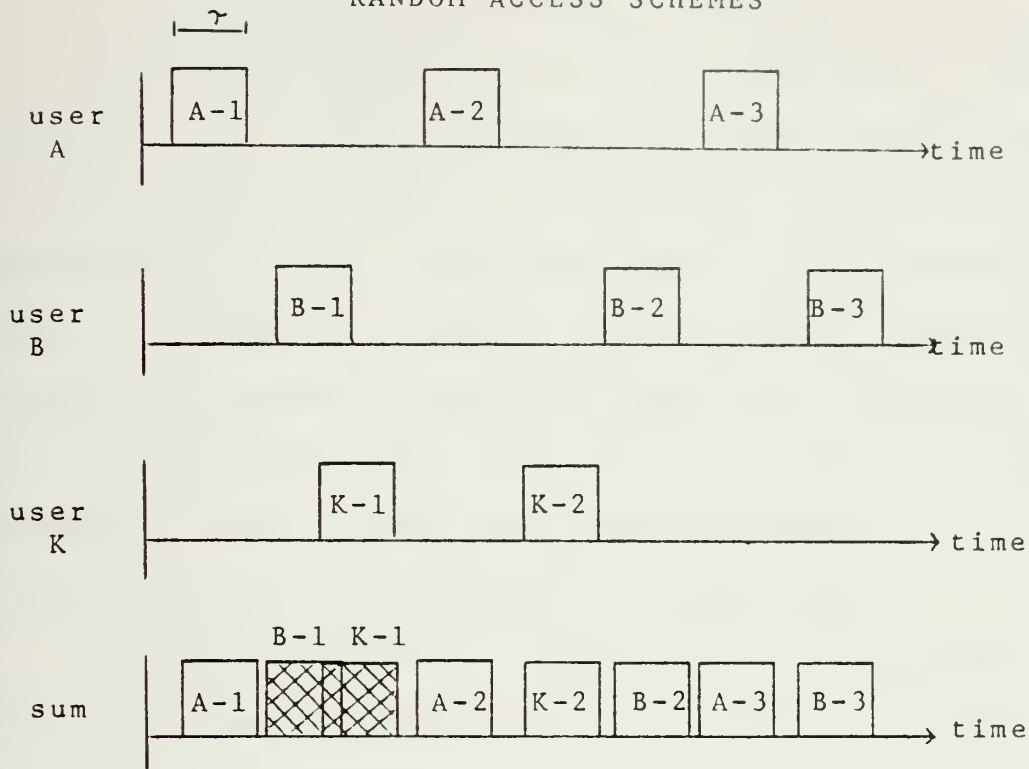


Figure 2

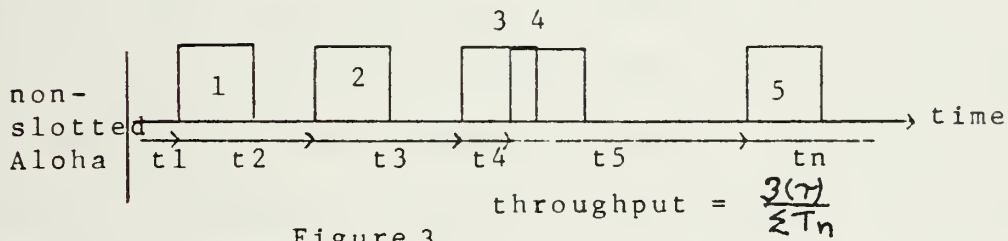


Figure 3

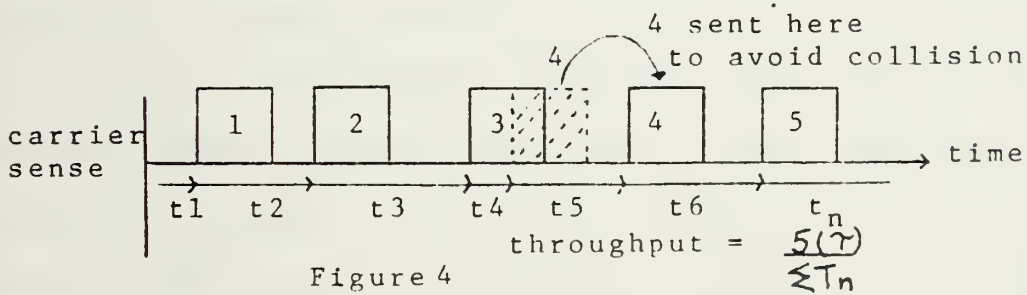


Figure 4

It should be obvious that throughput is higher in the carrier-sense mode due to the decreased number of collisions. In fact, analysis has shown that the maximum throughput possible with the non-slotted Aloha method is approximately $1/2e$ or 18.4%. Throughput in the slotted Aloha method is twice as high as the non-slotted method (36%). Throughputs as high as .80 to .90 have been obtained using the carrier-sense mode for cases in which the channel propagation time is small compared to message duration. Both of these methods are classified as asynchronous.

Another important measure of performance for distributed computer systems is a low delay time. This means that queries and responses between the system and the user take a minimum amount of waiting time. A short delay time is important when there are many interactive users on the net. Typically, in order to decrease delay time packet length is shortened. Long packet lengths, however, are necessary to increase throughput, shorten queue lengths and decrease processing overhead per bit.

On the battlefield, calls for fire from a forward observer would typically be short messages requiring a fast response, whereas, intelligence summaries are normally several pages long and require no reply. If both types of messages are carried by the same communications channel, one can readily understand why there are tradeoffs between throughput and delay time.

2. Category II

At the opposite extreme of categories of radio channel access methods, there are schemes which use completely static reservation access methods. These schemes pre-assign capacity to users and effectively create "dedicated" as opposed to multi-access channels. Such schemes as Time Division Multiple Access (TDMA), Frequency Division Multiple Access (FDMA), and Code Division Multiple Access (CDMA) fall into this category. The Joint Tactical Information Distribution System (JTIDS), Phase I, and the U.S. Marine Corps Position Location Reporting System (PLRS) use the TDMA scheme in which time is broken down into discrete intervals. The largest time cycle (called an "epoch" in JTIDS) is divided into thousands of smaller time slots or windows. Each user on the broadcast network is assigned one or many time slots in which he can transmit messages. After the passage of the cyclic time period his time slot again appears and he can transmit again. This technique, obviously, is highly dependent upon all users maintaining accurate time synchronization. The FDMA and CDMA schemes have also been developed to statically assign each user a fixed portion of the channel according to a unique frequency or code respectively. Here the problem is that a bursty user will often not use his preassigned capacity, in which case it is wasted. When a user is idle his portion of the channel cannot be used by other stations with traffic.

3. Category III

Between these two extremes are the dynamic reservation systems which only assign channel capacity to a user when he has data to send. In these schemes a certain portion of the channel is set aside in which to dynamically schedule transmission times. Several schemes fall into this category. In a Polling scheme the user waits passively to be asked if he has data to send. In an active reservation scheme the user asks for capacity when he needs it. In the Mini-Slotted Alternating Priority scheme a token is passed among numbered users in a prearranged sequence, giving each permission to transmit when he receives a token. The cost of these schemes is the overhead required to implement the dynamic reservations.

The following Figure 5. summarizes the costs of these three categories. [15]

The Cost of Distributed Resources

Access Method	Collisions	Control Overhead	Idle Capacity
Random Access Contention	yes	no	no
Dynamic Reservation	no	yes	no
Fixed Allocation	no	no	yes

H. CURRENT PROGRAMS

As was previously mentioned, before large scale distributed data systems can be introduced to a mobile battlefield, higher capacity communications hardware is required. The U.S. Army is working on two programs to meet this requirement. The two systems under development are the PLRS/JTIDS hybrid and packet radio.

An Army Letter of Agreement [25] which addresses the need for these systems reads: "There is an urgent need for communications capable of supporting existing and programmed automated systems for Air Defense, Field Artillery, Intelligence, and Command and Control. Characteristics of machine-to-machine communications, coupled with the need for fast reaction times and a high degree of mobility, result in a requirement for a specialized distributed data communications system. Without this data communications improvement, highly sophisticated and highly effective weapon systems fielded in the early 1980's will not operate to full potential."

In further describing present communications systems this document speaks of existing equipment as being "technologically old, generally manually connected, too large and immobile (multichannel), and requiring intensive maintenance and logistical support." "The current communication system cannot, without the addition of a digital data communications capability, meet the demand imposed by the emergence of automated systems in the early

1980's."

The precise extent of this communications shortfall is a matter of intense study by the Integrated Tactical Communications Study (INTACS) Update Study effort.

The PLRS/JTIDS hybrid offers a solution to this problem in the mid-1980's. Packet radio, also, is a promising technology to satisfy future tactical data distribution missions. Its development schedule, however, effectively eliminates it as a short term candidate.

1. PLRS/JTIDS Hybrid

A detailed description of the PLRS/JTIDS hybrid is not possible here. Briefly stated, however, the system is a computer based system which provides real time, secure data communications, and position location and reporting information for tactical forces. It combines desirable features of both the PLRS and the JTIDS systems, using modified equipment from both systems. This system is planned for introduction to the field by 1986.

In addition to work being done on the PLRS/JTIDS hybrid, the Army is monitoring or participating in some interesting tests with packet radio, described in the following sections.

2. San Francisco Bay Experimental Packet Radio Network

The current experimental packet radio network being supervised by SRI International is located in the San Francisco Bay area and has been operational since July, 1976. Figure 6 [16] shows a map of the network sites.

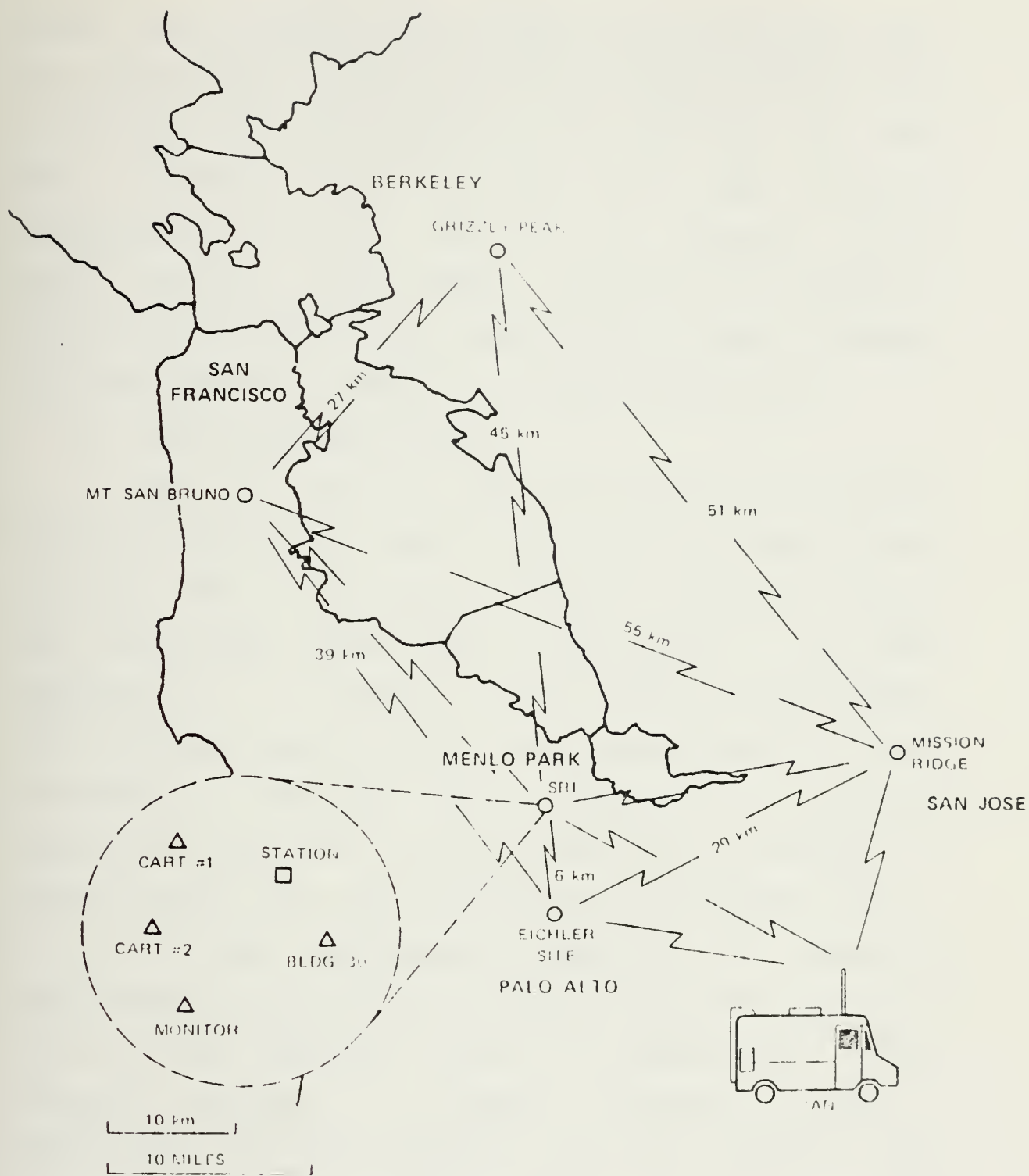


FIGURE 6 LOCATION OF MAJOR ELEMENTS OF THE PACKET RADIO TESTBED

There are two station PRU's located at the Menlo Park site. Each station has an associated PDP-11 computer attached. The network has four fixed repeater sites spread over the area and a variable number of packet radio user terminals (typically four to six). There are two vehicular packet radio terminals. These mobile terminals are an important aspect of the network, and a hand-off of a mobile terminal from one repeater to another is frequently exercised.

3. The Fort Bragg Test Bed

The U.S. Army has also recently set up a test bed for the evaluation of Packet Radio at Ft. Bragg, North Carolina. ARPA is sponsoring this effort, known as the Army Data Distribution System (ADDS). "The purpose of the ADDS experiment is to develop an environment in the resident XVIII Airborne Corps to permit user participation in the development, refinement and evaluation of innovative concepts for deployment of distributed data in support of future tactical Army data distribution requirements." [17]

In this multi-phased experiment, Ft. Bragg is experiencing a step-by-step build up of resources. The first phase of the experiment began in January, 1979. Three computer terminals, a network processor (called a TIP-Terminal Interface Processor) and a host computer were installed at Ft. Bragg and connected via commercial telephone lines into the ARPANET. After installation, operator training began in 1-2 day courses.

The second phase began in April, 1979 and the number of terminals was increased to fifty. The training in these phases acquainted operators of all ranks with the basic preprogrammed capabilities of the ARPANET including electronic mail, file management, directory maintenance, text editing, and printing. More specific applications geared to tactical information flow requirements are also being tested in garrison.

Phase III of the ADDS experiment is currently going on and involves the introduction of Packet Radio into the testbed. The PR network at Ft. Bragg will eventually grow to approximately 20 radios and 2 control stations. Initially, the radios are being employed in garrison to replace hard-wired connections. In the future, they will be deployed to the field in support of Corps exercises.

Packet Radio is expected to fulfill at least two major roles in the Ft. Bragg testbed. The first is to determine if this communications technology will satisfy the tactical data communications requirements of a corps on the battlefield. In this role, the system will be placed into the testbed as would any other communications system proposed for Army use. A second role is to provide a broadband communications channel for other systems under development which requires a data transfer capability. In this role the PRNET would provide communications for TACFIRE or some other intelligence or air defense system while maintaining its ARPANET connectivity as well. [17]

Reports from the Ft. Bragg testbed indicate that the XVIII Airborne Corps personnel have rapidly and enthusiastically adapted to the computer based communications technology. It is possible that the use of this data distribution technology can become as routine to commanders and staffs as voice communications are today.

[17]

In any case, the Ft. Bragg experiment represents an innovative and unique approach to investigating advanced concepts in Army doctrine and tactics. The testbed is a departure from traditional Army tests designed to arrive at production decisions, and is being driven by the urgent need for increased tactical data distribution capability.

I. FUTURE IMPLICATIONS

It is not an exaggeration to say that these are very crucial days for the U.S. military. Resource investment decisions are more important now than ever before in history, and the consequences of bad judgement offer potential for great loss. Some would argue that an ever-increasing dependency upon technology by the armed forces is a very dangerous trend. Nevertheless, it is apparent that electronics and telecommunications advances are beginning to have significant impact upon strategy. Chapter One of the U.S. Army Combat Communications Field Manual is entitled "Command, Control, and Communications (C3)". A quotation from this document helps to stress the intensity of future battles and the necessity of accurate

and realistic planning for that engagement. [24]

"The U.S. Army has arrived at a point where technology and reality have outrun our old tactics on fighting and left them in the dust. We have come to the shocking realization that the old way of doing things will not work any more."

"A good example of the change in combat reality facing today's soldier is an often used statistic from the Arab-Israeli War of 1973. In 20 days, over 1700 tanks were destroyed between the two sides. That's as many tanks as there are in five U.S. armored divisions. Technology has improved the weapons systems to the point where a tank has a 50-50 chance of being hit by the first round fired at it. We must retool our tactics to meet the reality of the next fight."

Certainly the capabilities of an army's command and control system has a great influence upon the capability of the force as a whole. The trends and technologies described in this chapter have far-reaching implications to doctrine and pose some problems that have yet to be solved. It is beyond the scope of this thesis to dwell on these implications in great length. A listing of the most obvious ones, however, is interesting and instructive.

1. Chain of Command

First of all, how will distributed systems change or affect the traditional chain of command structure? C3 system architectures typically reflect the chain of command within an organization, and this results in most C3 systems being

very hierarchical in structure. Distributed systems tend to be grid-like, peer structures. In a distributed system, is it advantageous to practice the concept of "skip echelon" reporting, in which certain levels of command may send traffic directly to the highest or lowest elements without intermediate "information only" stops? Certainly this would tend to decrease the time of delivery of messages and would avoid congestion on the network. But can the intermediate commanders afford to miss information that might prove to be critical or essential? What affect do distributed architectures have on the traditional role of communications centers and message centers? Would the requirement for these functions be eliminated? When the network is organized in a peer structure, how are protocols designed in order to preserve a "priority" system to message traffic? These are questions that remain to be answered.

2. Network Management

Next, a very important question is the manner in which network control and data base management is exercised. Although distributed systems appear outwardly to be decentralized, a static, inflexible network control design would make the so-called "distributed" system as vulnerable as its hierarchical predecessor. Clearly the responsibility for network management and data base updating must be transferable within the network, and the capability should be shared by more than one station. There is a tradeoff reached, however, between one station and multi-station

operation. The more nodes there are which can exercise network control, the more vulnerable the entire network becomes to spoofing and deception techniques, and the more costly and complex it becomes. It is interesting to note also, that some of the hardest decisions and longest delays in the JTIDS program have dealt with this question of network management.

3. How Much Redundancy?

In addition to these important questions one must ask also, "How much redundancy is enough?" Although distributed networks are more survivable and redundant, what kind of back-up systems are still required? Certainly an increased dependency upon satellite communications in today's world has decreased the investment in long-haul, high frequency (HF) communications systems. Some have argued with convincing reasoning that this is a dangerous position. It is generally more attractive to engineers and planners to invent and employ new systems rather than to improve the old. Nevertheless, it is necessary to retain and maintain older, proven hardware as back-up, secondary equipment. The question remains, "How much is enough?"

4. Propagation Loss Due to Higher Frequency

Because of the congestion existing at lower frequencies and the high bandwidth requirements of new automated systems, new communications equipment is being designed to operate at very high carrier frequencies. This restricts propagation to strictly line-of-sight distances.

While this situation is acceptable in ground-to-air and ground-to-satellite communications, it poses serious constraints upon ground-to-ground mobile communications, especially in rugged or forested terrain. The amount of propagation loss in this situation is highly significant, and the utility of these systems has yet to be proven.

5. Management Information Systems

It was mentioned previously that the development of an effective Management Information System (MIS) to integrate and display data to a commander and his staff is a formidable task. The Army, in fact, has been wrestling with this problem for over twenty years while attempting to field its Tactical Operations System (TOS, which is, in essence a MIS). After that amount of time, one questions whether such a system is nearer completion now than it was in 1960.

6. Interoperability

The military is fast finding out that interoperability means much more than mutually compatible equipment. It also means compatibility of procedures, software, and message formats. One should watch closely the continued development of JTIDS and attempt to judge the success of joint service programs. The interoperability requirement tends to introduce system complexity in an attempt to make the system "all things to all people." When one moves to the problem of interoperability in the NATO environment, the question again becomes, "How interoperable can equipment be without becoming too costly, too bulky and

generally ineffective?"

7. Voice vs. Data Circuits

Another important question yet to be resolved is the proper trade-off of voice and data circuits. Again, JTIDS can be the case in point. Some services seem to be side-stepping procurement commitments partly due to a lack of definition of system capability in this area. Although tactical commanders tend to prefer voice channels, voice channels require an enormously greater bandwidth allocation than data channels. If a commander is given the choice of having one voice channel or ten data channels into his command post, which will he choose? Which should he choose? What will the system offer?

8. System Cost

Finally, the question that will be asked most often is simply: "How much will the system cost?" Costs are divided into at least four categories. There are hardware and software costs, (it is common knowledge that the latter are now a greater consideration than the former) and there are initial procurement costs and life cycle costs.

Included in these costs is the manpower question. Is it realistic to think that as systems become more and more highly technical that the personnel who fix, operate and manage the systems will be better educated and more professionally competent?

IV. AN INTRODUCTION TO PETRI-NETS

A. INTRODUCTION

The purpose of this chapter is to introduce the reader to the particular modeling tool which forms the basis for this research. The history of Petri-Nets is first discussed. Then an explanation of how Petri-Nets work is presented. Following some simple examples, the chapter concludes with a brief summary of the strengths and weaknesses of this modeling tool.

B. HISTORY

The Petri-Net is named after its discoverer, Carl Adam Petri. These nets were developed in his early work in 1962 in Germany. They soon came to the attention of Anatol Holt who was then leading an Information Systems Theory Project for Applied Data Research, Inc. The work of this group eventually led to the theory of "systemics" [5] which dealt with the representation and subsequent analysis of systems and their behavior. At this point the modern formalism and notation of Petri-Nets was introduced. Holt also demonstrated the usefulness of the Petri-Net model in the representation of systems characterized by concurrent processes.

Perhaps the single largest source of research and literature regarding Petri-Nets has been Project MAC at the Massachusetts Institute of Technology. The Petri-Net model

was introduced to the researchers at Project MAC due to the association of Holt's group to J. Dennis' Computation Structures Group. This group has produced several Ph.D. thesis, together with many reports and technical memos dealing with Petri-Nets. In addition, MIT has sponsored two important conferences dealing with Petri-Nets. The first was the Project MAC Conference on Concurrent Systems and Parallel Computation held at Woods Hole in 1970. The second was the Conference on Petri-Nets and Related Methods, held at MIT in 1975.

This work, begun at MIT and continuing at other centers in the United States, until recently tended to concentrate on the formal or mathematical aspects of Petri-Nets. This work bears resemblance to the research in automata theory. It attempts to analyze systems by representing them as Petri-Nets, formally manipulating the representation in such a way as to derive information relating to the behavior of the modeled system. Because of the simplicity and power of Petri-Nets, they are excellent tools to use in the analysis of concurrent or asynchronous systems. They are finding their way into a number of diverse applications.

Petri, himself, is still actively researching, expanding his original theory. His extensions have led to a form of general systems theory called "net theory". Holt is continuing his research, concentrating on system representation and analysis of the formal representations.

[5]

C. HOW PETRI-NETS WORK

Simply stated, a Petri-Net is a model. More specifically, it is an abstract, formal model that analyzes the flow of information in systems. [5,19] Petri-Nets also describe not only the information flow, but the controls and constraints of such flow. A Petri-Net graph models the static structure of a system in much the same manner as a flowchart models the structure of a computer program. As a modeling tool, Petri-Nets are especially useful in modeling systems that exhibit asynchronous and concurrent activity.

A Petri-Net consists of a collection of "events" and "conditions." In graphical notation, conditions are conventionally represented by circles and events are represented by bars. The Petri-Net is given structure and the capacity for interaction by connecting events and conditions with arrows.

An arrow from a condition to an event signifies an input condition to that event and implies that every occurrence of the event terminates the "holding" of that condition. An arrow from an event to a condition signifies an output condition, and in this case, the occurrence of the event commences the holding of the output condition. The graphic notation for a condition which holds is the marking of that condition by a "token".

The behavior of a system may be thought of as the occurrence of events as time progresses. If all input conditions to an event hold, the event can occur. This

results in the holding of the event's output conditions.

Conditions may also be called "places" or "nodes" and events may also be referred to as "transitions." In the Petri-Net model of a system, directed arcs connect places to transitions and transitions to places.

The Petri-Net is first given a particular structure of places and transitions, and then it is "executed." Execution is governed by a "firing" protocol.

Simply stated, a transition may "fire" (symbolizing the occurrence of an event) when all input conditions or places into that transition are marked with a token. When all inputs into a transition are marked, the transition is said to be "enabled". Figure 7 shows an "enabled" transition.

Execution of the Petri-Net involves the cyclic checking of all transitions once during each time interval. Each transition that is found to be enabled is fired, and tokens are moved from the input places of the enabled transition to the output places of that transition. This procedure continues for a set number of iterations. The flow of tokens in the Petri-Net thus symbolizes the flow of information or control in the modeled system.

By devising special methods for marking the number of tokens at the Petri-Nets nodes, the system status can be accurately and effectively recorded. The state of the system is reflected by an ordered set of mark status indicators which correspond to the nodes of the graph structure.

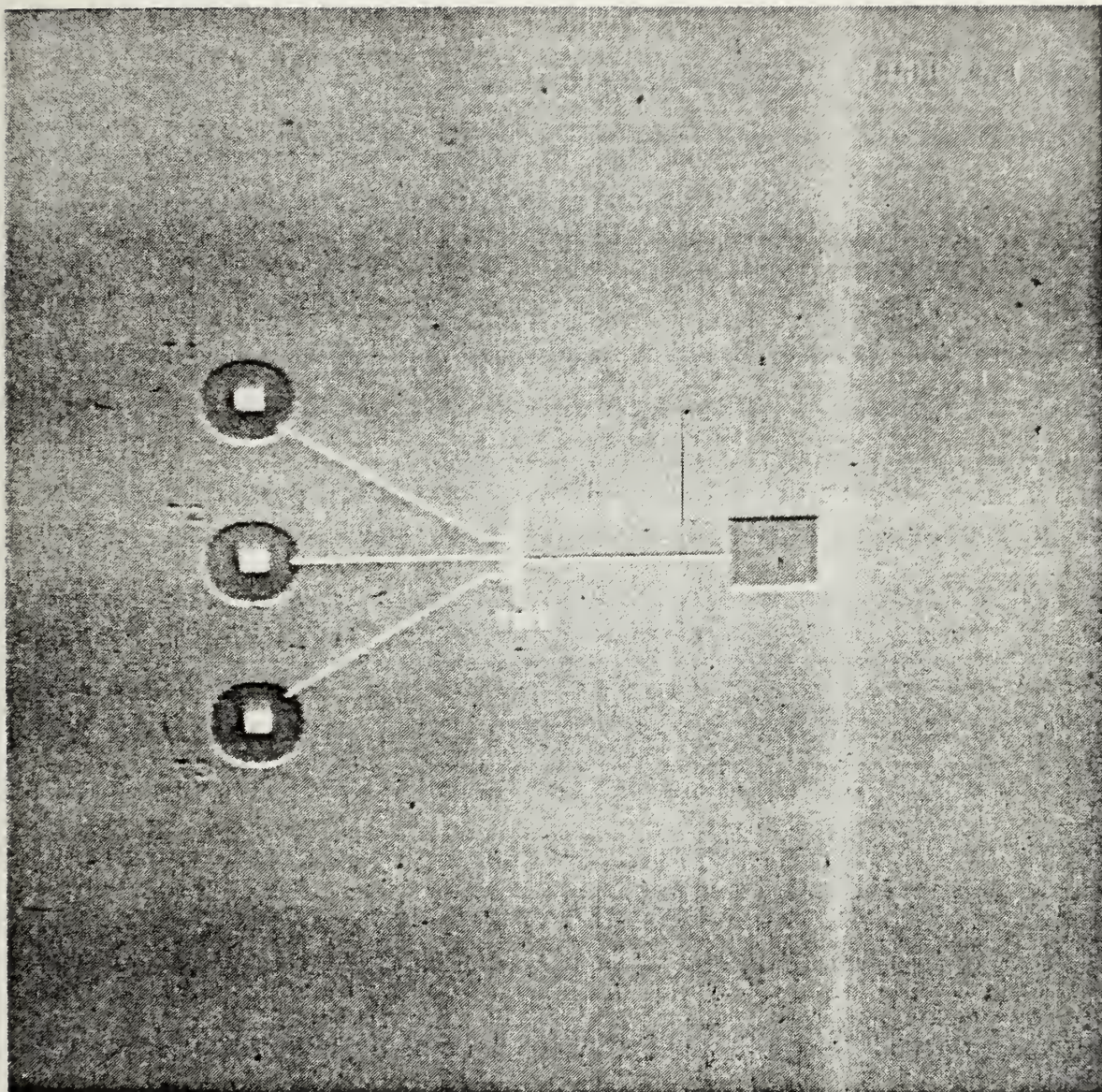


Figure 7.

In this thesis such an effective marking method is uniquely employed to give the viewer an accurate snapshot of network status.

Petri-Nets have rapidly gained acceptance over the last decade. Along with this acceptance has been the furthering of the understanding of Petri-Net properties.

D. A SIMPLE EXAMPLE

Figures 8 through 11 show the various states of a simple Petri-Net as execution occurs during four successive time intervals. [21] Notice four places (P1, P2, P3, and P4) and three transitions (TR1, TR2, and TR3). The directed arcs denote the interaction and relationships between input and output conditions. For instance, TR1 will become enabled when P1 (its only input condition) is marked with a token. At the time that TR1 fires, the token will be removed from its input condition (P1) and placed in its two output conditions (P2 and P4). In this manner flow of information or control is followed through the modeled system. Figure 8 shows the network at time = 0 with one token placed in P1. Figures 9 through 11 depict the Petri-Net as it continues execution through time = 3.

E. ADVANTAGES AND DISADVANTAGES

The following characteristics of Petri-Nets were found by the authors to be strengths when using this particular modeling tool for simulations in the context of this research:

A MARKED PETRI-NET (TIME = 0)

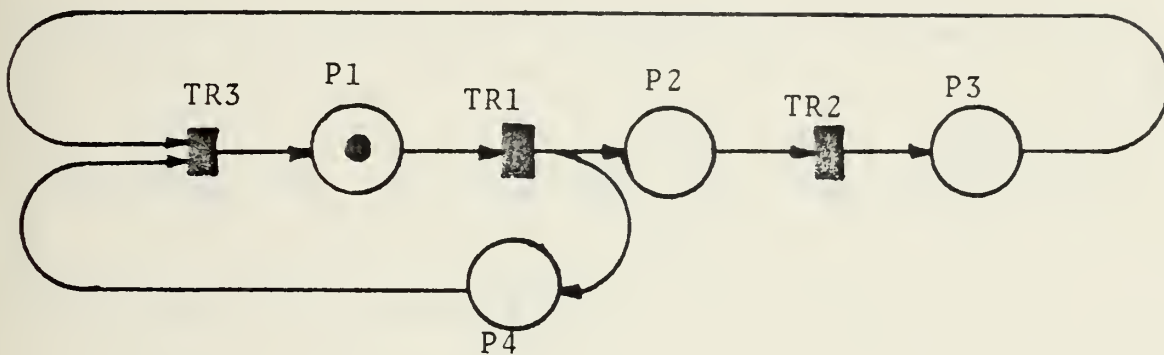
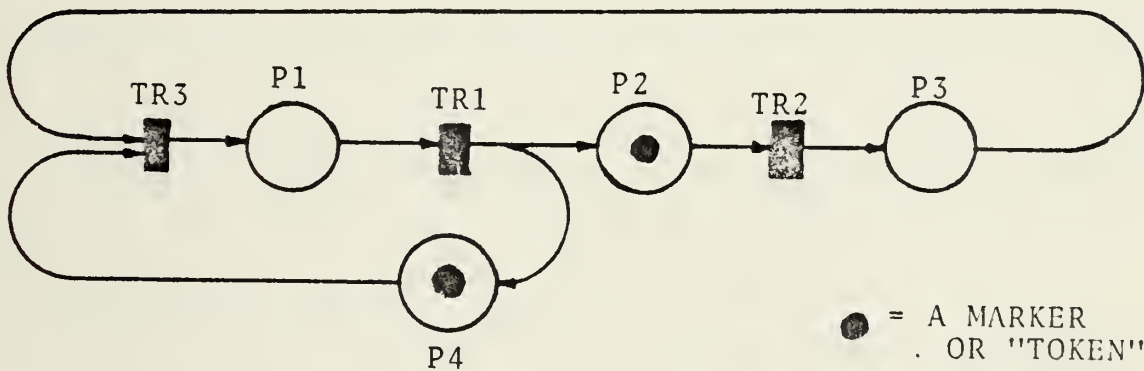


Figure 8

A MARKED PETRI-NET (TIME = 1)



● = A MARKER
OR "TOKEN"

Figure 9

A MARKED PETRI-NET (TIME=2)

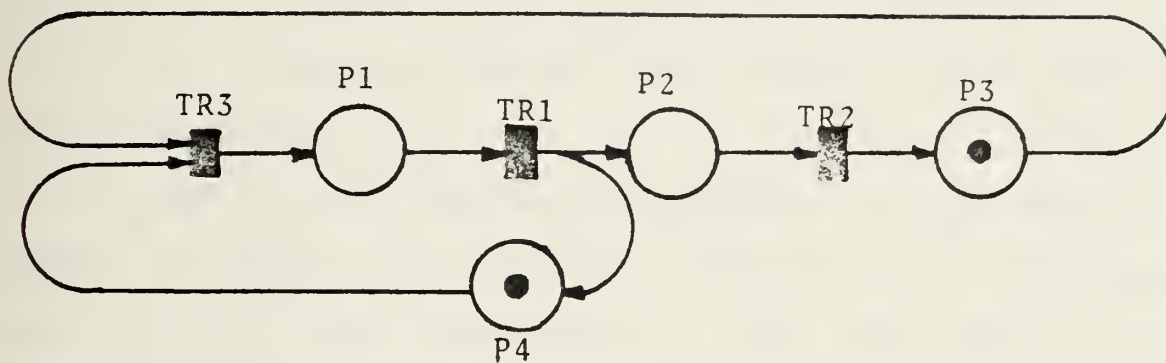


Figure 10

A MARKED PETRI-NET (TIME = 3)

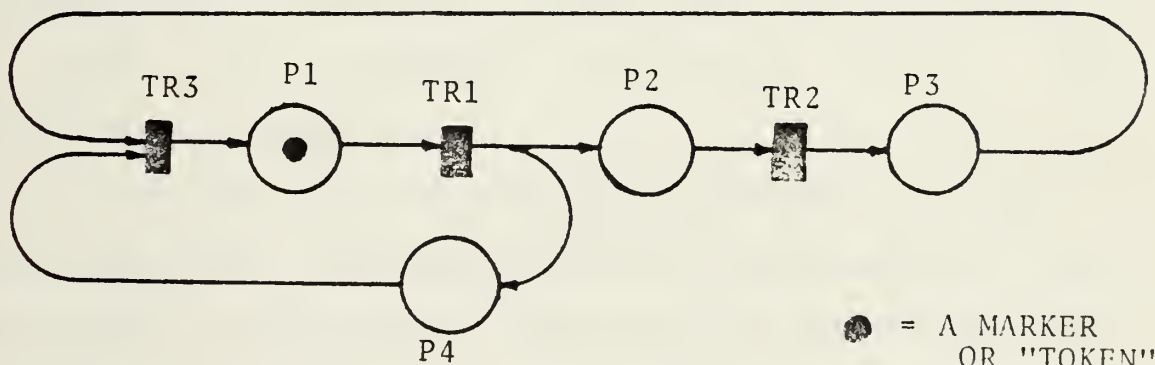


Figure 11

1. The rules governing Petri-Net execution are simple and easy to understand. This methodology can be quickly grasped by those with non-technical backgrounds who would ordinarily be unable to understand mathematical or analytical modeling. Yet Petri-Nets retain a high degree of precision and accuracy.

2. There is much flexibility inherent in the Petri-Net graph to model wide ranges of complexity. For instance, a model can be further abstracted by the replacement of a complex network of nodes by a single node.

3. There is a large degree of flexibility in assigning time intervals during execution.

4. Petri-Nets are well-suited to "snapshot" portrayal of network states. This advantage is important in simulation languages and is considered a strong point of languages such as GPSS and SIMSCRIPT. Petri-Nets possess this characteristic by nature.

5. Petri-Nets have the potential for a wide variety of uses. Basically, any process that can be flow-charted could be expressed by Petri-Nets. Applications could include: flow of information or control in an organization, information flow in electronics hardware, representation of computer software or, procedures and stages of development in a management program.

6. Petri-Nets lend themselves well to computer graphics display.

7. Petri-Nets are very effective when modeling concurrent, asynchronous activity in a network or system.

Certain weaknesses also became apparent to the authors in the course of this work. They are listed as follows:

1. Although Petri-Nets are basically simple to understand, the small building blocks of a network soon become exceedingly large and complex when large systems are modeled. The input files to some of the larger simulations in this paper were more than 1400 lines long. These networks must be drawn on paper before their entry into the computer, and this kind of effort soon becomes very tedious and prone to error.

2. Petri-Nets are best suited to concurrent, asynchronous behavior. When non-concurrent, synchronous behavior in a system is modeled the Petri-Net assumes a large amount of overhead.

3. The fact that Petri-Nets are not generally well known in the computer communications community could be a disadvantage when the user wishes to prove the accuracy of his model.

4. The fact that the simulator employed in this thesis effort was deterministic could be considered either as an advantage or disadvantage depending upon the application. Many simulations have value because of their stochastic nature. Certain classes of experiments, however, need to be understood not because of "chance" happenings but because of the operation of the "laws" of nature working upon the

elements of the experiment.

As with any modeling technique, success is achieved through the modeler's familiarity with the modeling tool. Petri-Nets provide an excellent means to model those applications best characterized by their asynchronous, concurrent nature.

V. PETRI-NET SIMULATIONS OF COMPUTER COMMUNICATION NETWORKS

A. INTRODUCTION

Once the basic rules of Petri-Net execution are understood, it is a simple matter to apply these rules to a communications network. The system modeled is the network itself. The movement of tokens in the Petri-Net represent the flow of information within the system: in this case, message traffic in the network. Each token becomes a discrete amount of information contained within the message.

The places in the Petri-Net graph are used to represent communications nodes within the network. The directed arcs of the Petri-Net graph are used to represent the communications links or channels which interconnect the nodes. The transitions between nodes indicate the availability of the channel to carry traffic. If the transition is enabled, the channel is clear, and the message is relayed from input node to output node.

The careful structuring of the Petri-Net graph imposes upon the modeled system a variety of network protocols. An advantage of using Petri-Nets for simulation purposes is that the logic and protocol of the system are entirely contained in the structure of the Petri-Net graph rather than in a complicated mathematical algorithm within simulation software.

B. TYPES OF NODES

The graphical output of the simulation attaches significance to the shape of the nodes displayed to the screen to facilitate recognition and interpretation. The experimental packet radio network in the San Francisco Bay area defines three primary types of nodes: terminals, stations, and repeaters. A terminal is a user node at which traffic is inputted or to which traffic is destined. It could be a fully automatic sensor, a handheld device, or a keyboard with CRT; but a terminal is a place where users connect to the network. The station is the node at which network control is exercised. The station typically keeps network statistics, monitors flow and congestion control, assigns routing, and performs data base management for the system. It is a terminal with increased processing capability usually provided by an attached mini-computer. The repeaters are stand-alone devices placed in numerous, dispersed positions throughout the network to act as relay sites. Repeaters do not act as origins or destinations of traffic, but they serve the purpose of extending the geographical range of the network beyond a typical line-of-sight distance.

While all networks do not use this identical terminology, these three functional nodes summarize the requirements of communication hardware in most networks. In the simulations in this thesis, three types of figures and labels represent the functions of nodes as described above.

C. A SIMPLE APPLICATION EXAMPLE

In order to understand more completely the application of Petri-Nets to communications circuits, refer to Figure 12. In this diagram two distinct one-way communications channels are represented. The first goes from T1 to T2 and the second goes from T3 to T5 and through T4. The individual tokens are representative of packets of information in a packet switched environment. The three additional nodes forming a triangle in the center of the diagram impose a special firing order upon the transitions in the communications channel. These center nodes are systems overhead which ensure that only one "packet" can be transmitted during a single time frame. In fact, if the three additional center nodes are thought of as a clock, then the entire network is a representation of Time Division Multiple Access in a network. A terminal can only transmit during a particular assigned time slot. After the time slot passes, the user must wait until the clock cycles back to his slot again. Because each transmitter has a unique time slot assignment, no two terminals can transmit during the same time, and collisions of packets on the radio broadcast channel are eliminated.

This explanation should give the reader a simple idea of the manner in which various protocols are represented. Obviously the Petri-Net in Figure 12 allows only non-concurrent activity on the communications channels.

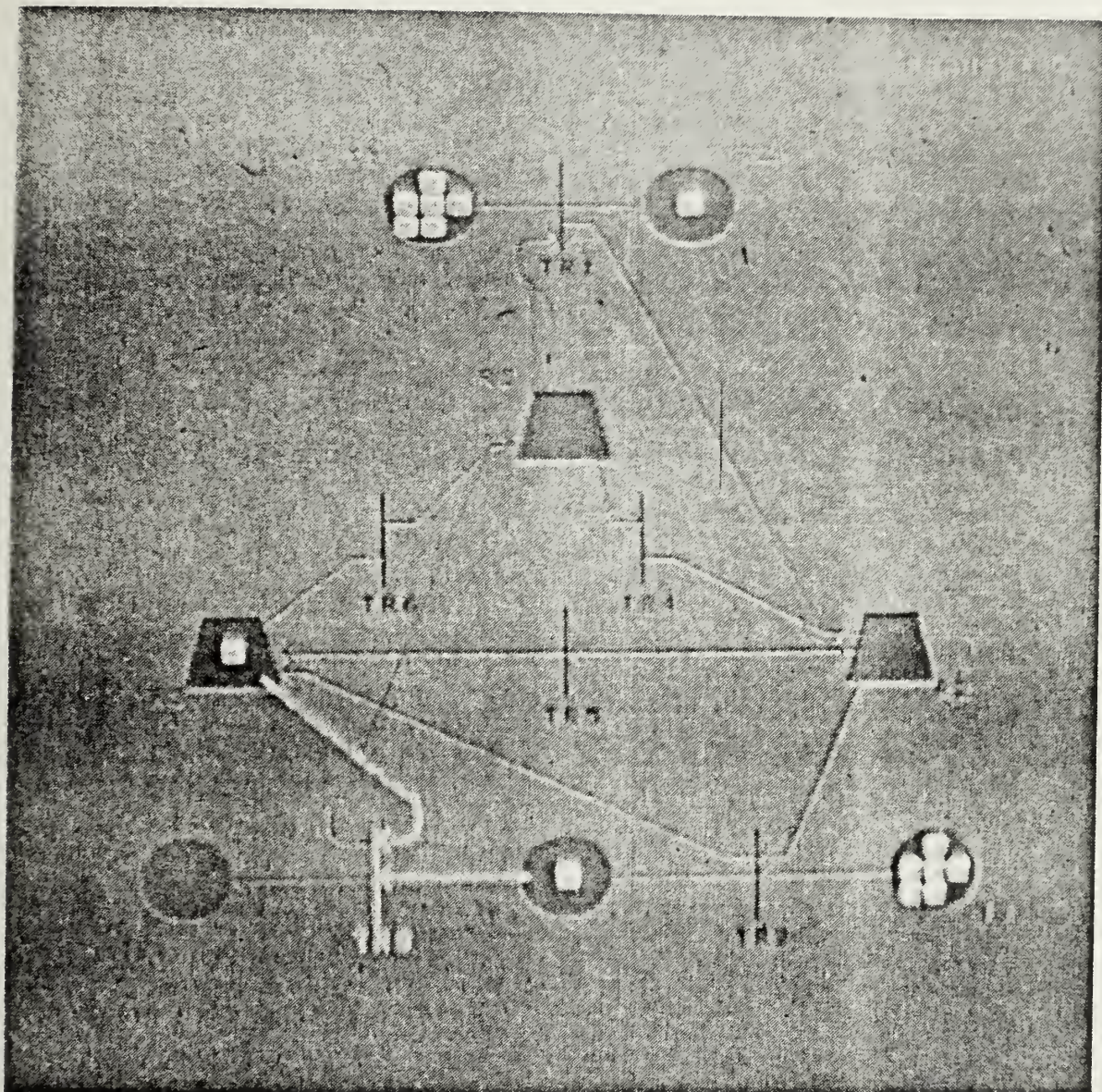


Figure 12.

The removal of the additional overhead would allow concurrent activity. In fact, this is sometimes desirable. For instance, those networks such as AUTODIN II which will make use of leased landlines will allow concurrent activity in the network. The Petri-Net is more efficient when modeling concurrent activity. The requirements to ensure non-concurrency as in Figure 12 , causes one additional node and one additional transition to be placed in the Petri-Net graph for every transition on the communication links. Using the fundamental Petri-Net described here as a small building block, a system of considerable complexity can be built with many origin and destination terminals, and which allows packets to flow two directions with multiple, alternate paths from source to destination.

D. RANDOMNESS IN PETRI-NETS

The Petri-Net simulator used in this thesis work is deterministic. After the simulator begins execution there is no means to interactively alter the sequence of events, and there is no element of randomness within the simulator. Because of this situation, the same input file will always give identical output. Although the capacity for alternate routing within the communications network is implemented, the tokens do not randomly "choose" their routes during execution. They can only follow their pre-assigned routes from origin to destination. This means effectively that fixed routing instead of adaptive routing is represented in the simulations. This is not necessarily a disadvantage,

however, as some simulations have shown better throughput and time of delivery results using fixed routing over adaptive routing [20].

It might be advantageous to modify the Petri-Net simulator to make it more stochastic in character. This might be accomplished in at least two ways. First, the initial marking of the network could be varied randomly at the beginning of each run. Certain key nodes could be marked or left unmarked according to the result of a call to some random number generator. Then the initial random state of the network would affect the end result of the output file.

A second way to add randomness deals with the Petri-Net concept of "dynamic conflicts" (see Figure 13, for an example of this particular network state). In this figure, the reader will notice that both Transitions 1 and 2 are enabled, but both cannot fire. Only one transition can fire, since in so doing it removes the token from T2 and disables the other transition. Thus TR1 and TR2 are said to be in conflict. This basic relationship can be used to create either deterministic or nondeterministic behavior. If the Petri-Net simulator is deterministic, the firing order for transitions in conflict is fixed according to a certain rule. This describes the case in this paper. The firing order of transitions is explicitly defined by their ordering in the input file. In the case of Figure 13, if TR1 is listed in the input file before TR2, and if a dynamic conflict occurs between them, TR1 will always fire first and

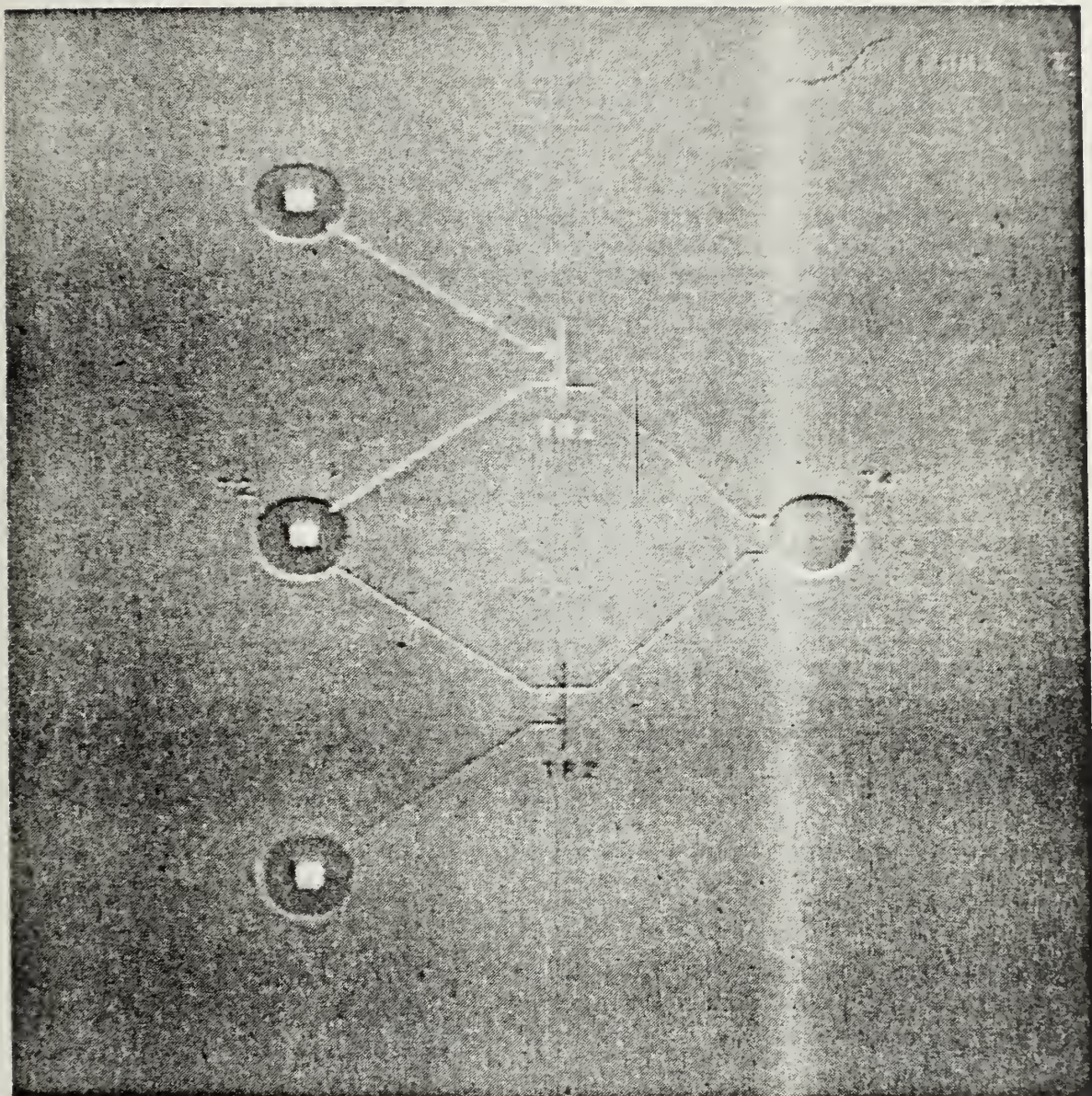


Figure 13.

TR2 will not fire during that time interval because it will have been disabled. This set of rules gives a strict priority of firing to the network.

The firing order of transitions in conflict could be modified so that it occurred in a random fashion. This would allow the Petri-Net to be executed in a non-deterministic manner and add the missing aspect of randomness to the outcome if so desired. In fact, Petri's first networks were non-deterministic because of this factor.

E. MEMORY STORAGE REPRESENTATION

The amount of memory storage in any particular communications node is also easily represented in the simulation. In this work a maximum number of seven packets is allowed at any one node. This number reflects the buffer size of seven packets in packet radio technology. If at any time a node accumulates more than seven packets, the buffer size has been exceeded and packets would theoretically be lost. When the buffer space is exceeded, the number of overflow packets is displayed outside the node in a red warning color.

F. SYSTEM LOAD AVERAGE

The system load on the network can be represented by the number of packets in transit at any one time interval. The system load then can be varied by controlling the frequency of message generation. The shorter the interarrival time between generated messages, the busier the network will be.

At some point the network will be saturated and unacceptably congested if message input is greater than message throughput.

It is a simple matter to construct a message generator using Petri-Nets. See Figure 14. In this figure Transition 1 is firing to two outputs. One can be thought of as external and one output, R1, can be considered internal to the network. The external node can represent entry into the communications channel. The internal output feeds a token back into the "generator" and will, therefore, fire other packets into the network at regular intervals. The other repeaters (R2, R3) can be thought of as delays which slow down the frequency of message generation. This configuration of places and transitions constitute a message generator. The frequency of generation can be staggered and then several generators may be placed at the input of every communications circuit. In this manner the system load on the network may be varied.

G. TIME REPRESENTATION

The Petri-net model is very good at representing the net status at distinct time periods. In fact, each time interval displayed to the screen gives an excellent "snapshot" of network status. This is an important advantage inherent to the Petri-Net simulation. Another advantage is the flexibility afforded in assigning the time interval. The user has the prerogative of making each time interval as long or as short as is necessary. Successive snapshots may

represent the passage of time of 1 millisecond, 100 milliseconds, or 1 hour, depending upon the application and network being modeled.

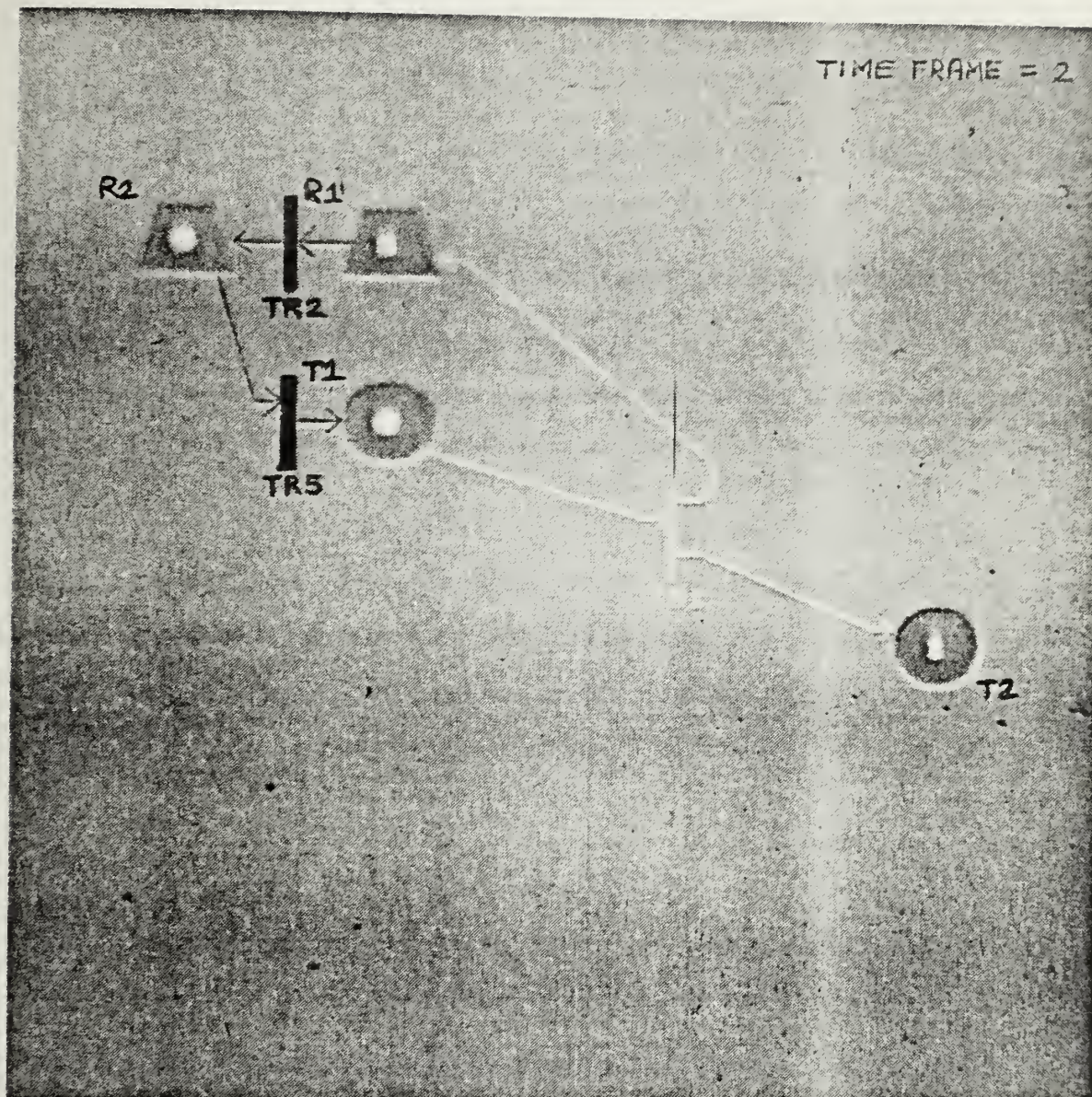


Figure 14.

VI. THE EXPERIMENT

A. DESCRIPTION

An experiment was designed to demonstrate the feasibility of using the Petri-Net simulator as a predictor of network performance. The experiment was performed by keeping certain parameters constant and varying others. A series of six input files were run through the simulator. In all six files the network architecture and the fixed routes were kept constant. Figure 15 shows the network. There were five origins of message traffic and five destinations. Four nodes were designated as terminals plus one station. There were four repeaters that performed relay functions within the network. Each of the five origins could send traffic to one of four destinations. Each source to destination combination had two routes for traffic to take. This made a total of 38 possible fixed routes in the network.

The controlled variables for the experiment were system load, concurrent vs. non-concurrent activity, and polling frequency of various circuits. The first three runs of the experiment were done with a high load. For the second three runs the message generators were slowed down to give a low system load. Some input files allowed concurrent network activity and some required non-concurrent activity. On two of the non-current runs the frequency of polling certain selected circuits was increased. This would represent the

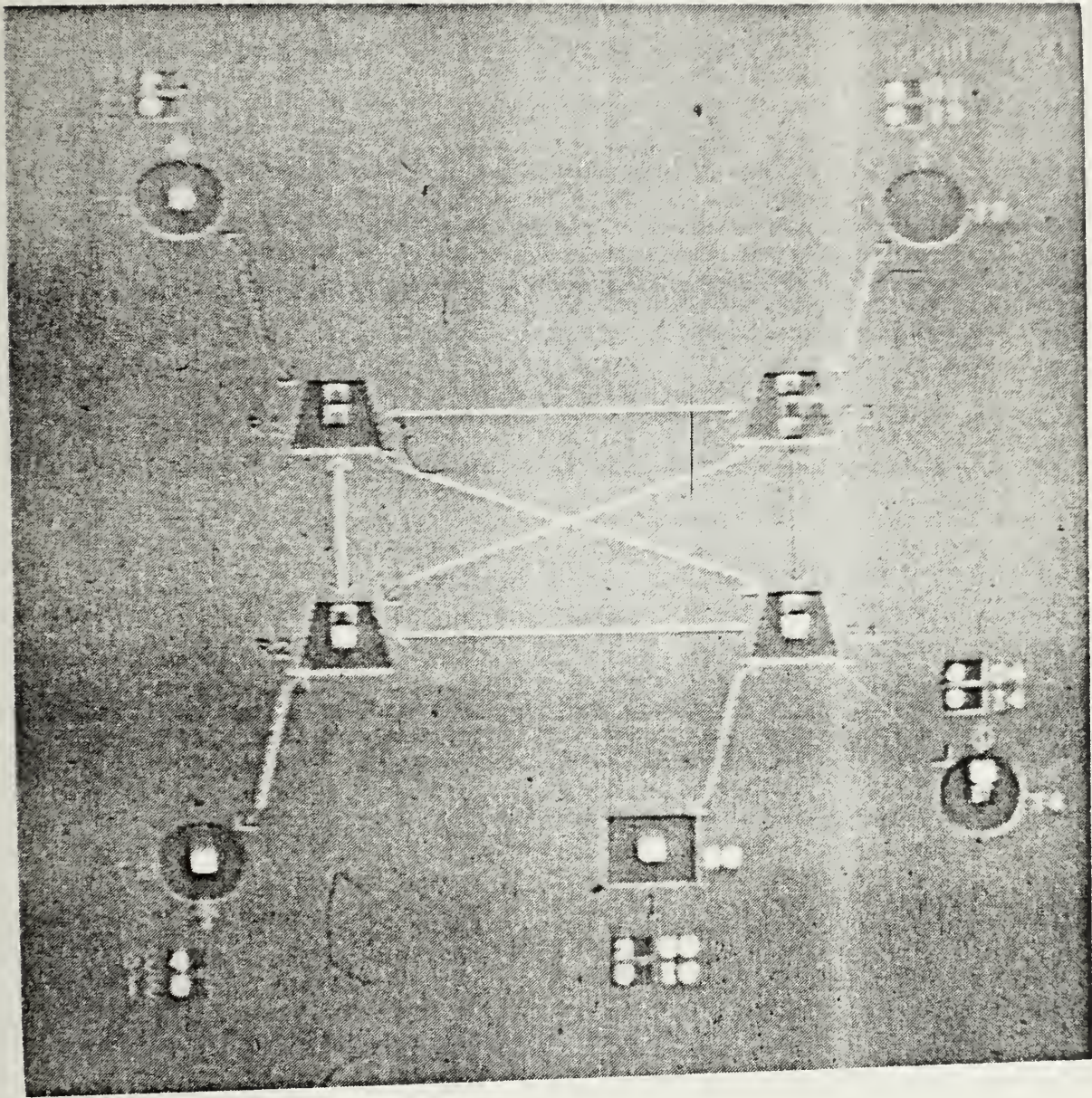


Figure 15.

equivalent of assigning a certain user more time slots in a TDMA scheme than another. It would give priority to those subscribers who have more traffic to send.

The only statistic gathered from the experiment was throughput measured in the number of packets which successfully reached their destination. This number could be extracted directly from the output queues at each terminal or station. The following points summarize the characteristics of each of the six input files:

1. Runs 22-24 were run at high load.
2. Runs 25-27 were run at low load.
3. Runs 22 and 25 exhibit concurrent network activity.
4. Runs 23 and 26 exhibit non-concurrent network activity with equal polling frequency of all circuits.
5. Runs 24 and 27 exhibit non-concurrent network activity with weighted polling on certain circuits that are terminated at T1.

B. RESULTS

Figure 16 shows a summary of throughput statistics from the experiment. Certain results are no doubt obvious, but the quantitative nature of output statistics validates prior assumptions.

The following observations are noteworthy;

1. Allowing concurrent activity on the network greatly increases throughput. The number of packets successfully transmitted in RUNS 22 and 25 was on the order of four times larger than the non-current runs. This question of

THROUGHPUT IN NUMBER OF PACKETS

HIGH LOAD

	RUN22	PUN23	RUN24
T1	92	16	25
T2	62	16	15
T3	77	18	16
T4	58	18	17
S1	<u>62</u>	<u>17</u>	<u>16</u>
TOTAL	352	85	89

LOW LOAD

	RUN25	RUN26	RUN27
T1	36	17	19
T2	38	17	16
T3	35	18	16
T4	33	16	15
S1	<u>31</u>	<u>17</u>	<u>16</u>
TOTAL	173	85	82

Figure 16.

concurrency has significant implications to radio broadcast systems. For instance, what would be the advantages gained in the Packet Radio Network if terminal radio output power were reduced so that the terminal could only talk to its nearest neighbor rather than the entire network? This situation could allow simultaneous transmission of packets without the threat of collision. Also, a multiplexing scheme within the network could allow concurrent activity. These types of considerations could be modeled easily with the appropriate modifications to the Petri-Net graph.

2. Increasing the frequency of polling on selected circuits increases the throughput of those circuits. The reader should note the number of packets received at T1 in RUNS 23, 24, 26, and 27. RUNS 24 and 27 show a slight increase because certain circuits destined for terminal T1 were polled more frequently. This situation could easily reflect the assignment of more time slots to certain priority subscribers in a TDMA scheme. Again this modification was performed simply by restructuring the input file to the Petri-Net simulator.

3. The reader will note that the total throughput in the non-concurrent runs is largely the same regardless of the high load - low load factor. This is because the system is basically "saturated" during non-concurrent activity at both high and low loading. Although the user might desire to improve throughput by generating more messages and trying to force them into the system, the network will give the same

results because it is already operating at full capacity.

4. A visual inspection of the output as PIN22 executes reveals that the system is essentially operating at peak efficiency under maximum load. The viewer will notice that the buffers at every location are frequently filled to capacity but seldom are overflowing.

VII. RECOMMENDATIONS AND CONCLUSIONS

A. RECOMMENDATIONS

The potential exists for significant follow-on work to this thesis. Major topics for future work include the following:

1. The development of a language to describe networks, and the inclusion in the software of a "front-end" program that would make the input file less cumbersome. The program could be static, like a compiler, or interactive, written to query the user about a number of basic network parameters. For instance: How many nodes do you desire in the network? What are the paths of traffic from source to destination? Do you wish to allow concurrent or non-current activity? The software would take the user responses to such parameters and construct the input file from the responses.

2. A statistics gathering package should be written to collect and collate vital network parameters as the simulator is executed. Such a package would keep a running total of such items as:

- a. average number of packets at a node
- b. average number of packets on a circuit
- c. percent use of a circuit
- d. average time delay between transmission and reception along each fixed route
- e. number of messages lost

f. number of messages which successfully reach destination.

These statistics could be formatted and displayed by a post-processor.

3. As previously described, the addition of randomness to the simulator might be considered desirable, and , if so, the proper modifications could be added.

4. As the system approaches "production status", the entire question of model validation needs to be addressed carefully. Much could and indeed has been written on the subject of how to demonstrate your simulation is accurate. [10] As in most simulations there is a tradeoff between the degree of complexity represented in the model versus the largeness of the network. If a network is small or if only a small segment of the network is modeled, the degree of detail can be great. If the network is very large, however, the data storage capability might not allow the same level of detail.

There were two constraints to validation posed in this work. First, the memory capacity of the PDP-11/70 was stretched to the limit on the larger runs. The mini-computer offers 128,000 bytes of memory which are partitioned into thirds. This gives any one user $128/3$ or 42.6 K bytes of memory. The graphics programs approached and then exceeded this bound before the work was completed. This forced the division of one program into two separate programs of 38K each. The file sizes for the output files from the Petri-Net

simulator are also very large. The larger of the input files produced output files on the order of 300K bytes. Figure 17. shows the relationship between number of nodes in the Petri-Net versus the size of the output files. In order to make claims of simulation accuracy based on real world networks, more memory is needed.

It should be added at this point that limitation of memory caused changes in the graphics display programs and in the overall organization of the software. The potential exists in the actual Petri-Net simulator (written in fortran and discussed in Appendix A) for the execution of Petri-Nets of well over a thousand places. If the user does not require a graphics output from this software package, then larger networks can be simulated on the PIP 11/70.

Secondly, the networks that were of most interest to the authors are largely experimental, unproven technologies. The Autodin II network is not yet operational and no statistics are available for validation purposes. The JTIDS technology is likewise not established operationally and many of the system characteristics are classified. Packet radio, which formed a good deal of background for the simulation, is still in its infancy. Also, packet radio employs a random channel access scheme which is contrary to the deterministic nature of the Petri-Net simulator.

Because these are new technologies, new routing and flow control algorithms and a host of different kinds of protocol are presently being developed. It is difficult to model a

system that has not reached its production state. Rather than being concerned about modeling a particular network that is still in a process of change and then trying to prove the simulation valid, it seemed wiser to leave the models in a more general state. By executing a variety of different input files, the simulations demonstrate the feasibility of future validation.

B. CONCLUSIONS

The conclusions of the authors are fourfold:

1. First, computer communication networks can be meaningfully modeled with the use of Petri-Nets. The background research to this thesis discovered no previous work which employs Petri-nets in the manner described in this paper.

2. Secondly, Petri-Net models of networks can be executed and displayed effectively on a color graphics terminal. The results of such a simulation are more easily understood than the common, hard-copy outputs produced by most analytical or queuing theory simulations. The color graphics output also could have considerable educational value. Again, background research uncovered no instance in which Petri-Nets were displayed and executed on a color graphics terminal.

3. Thirdly, and perhaps most importantly, the implementation of such a modeling technique in a production environment as a predictor of system performance appears feasible.

4. Fourthly, there appears to be considerable benefits to encouraging future, carry-on work in the subject matter of this thesis.

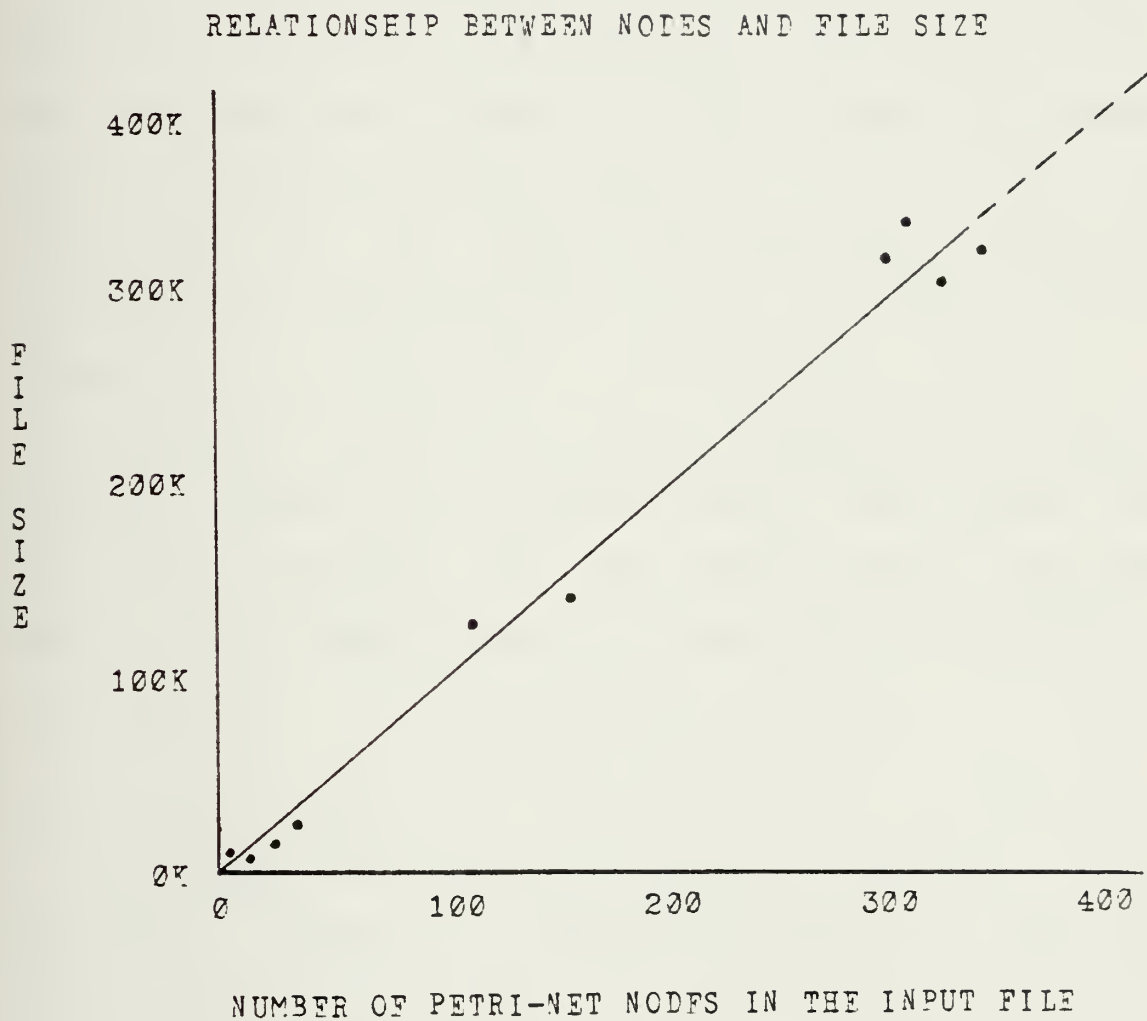


Figure 17.

APPENDIX A - USER INSTRUCTIONS FOR THE PETRI-NET SOFTWARE

A. INTRODUCTION

This chapter is written to describe certain procedures and syntax peculiar to the simulation software. Assuming that an interested student or faculty member is somewhat familiar with the theory and structure of Petri-Nets, the instructions in this section will allow him to apply the simulation and graphics output to his particular modeling problem.

Figure 18 shows the various components of the entire software package, the program source code sizes, the programming language, and the output files. The reader should refer to this figure as he reads the instructions in this appendix.

B. THE INPUT FILE

The input file written by the user contains all of the information necessary to uniquely describe the Petri-Net model. This file is read by the fortran f4p program named, "simulator". The input file must be named "RUNYX, where XX can be any number from 01 through 99. There are three main divisions of the file,

- Part I Places
- Part II Transitions
- Part III Marking

SOFTWARE METHODOLOGY

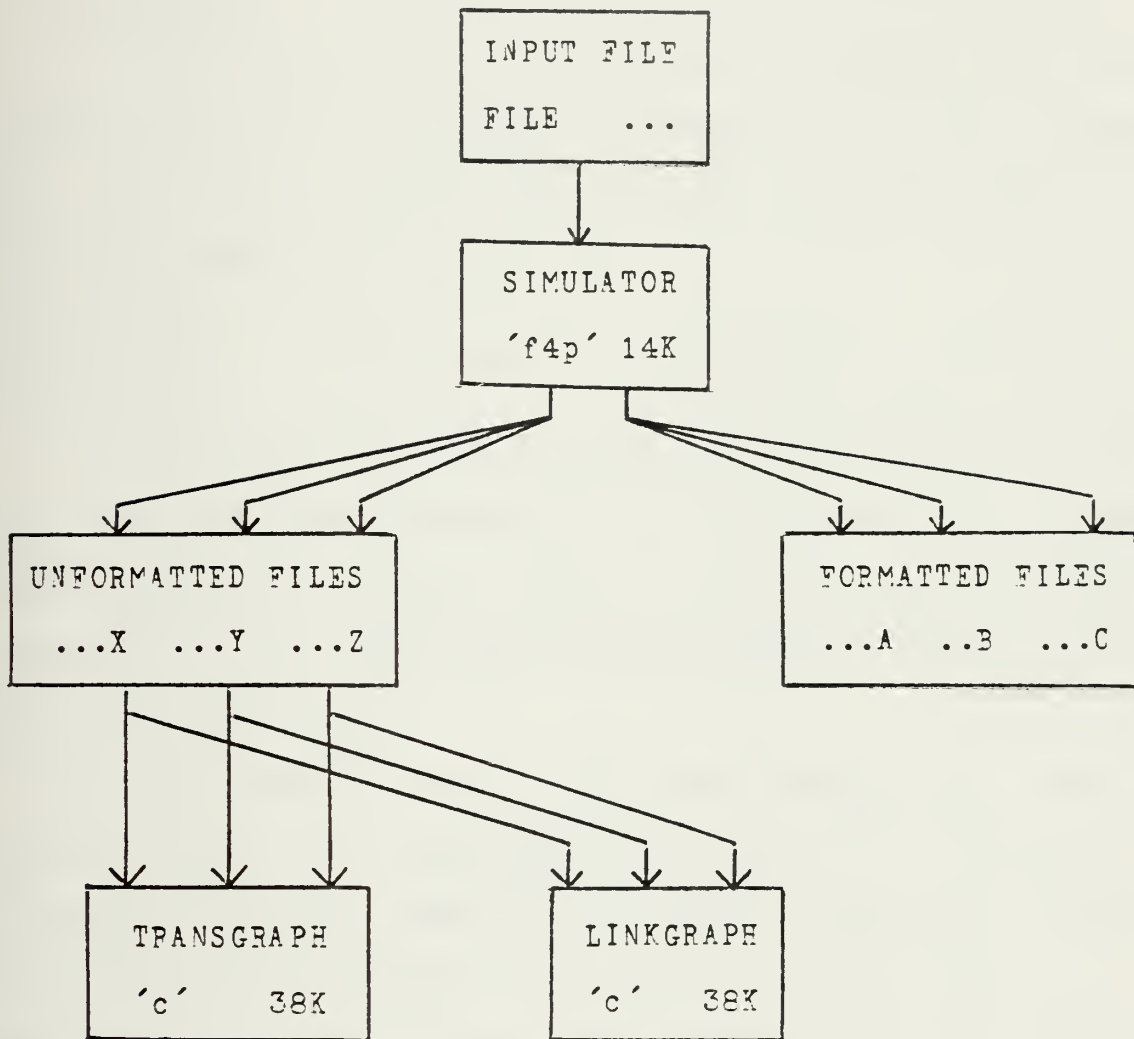


Figure 18.

Appendix B. shows a sample input file. The reader should refer to this example as he reads the following instructions.

1. Places

The first line of the file specifies the total number of places to be read in. Beginning on line number two, the names and locations of the places are listed. The following format applies:

Place Name X-coordinate Y-coordinate Plot Flag

The place name must be less than 13 characters long. Only the first two characters will be displayed as a label on the graphics terminal. There are unique instructions for labeling places in the multi-routing version. These will be discussed later.

The first letter of the name specifies the type of figure that will be displayed. The letter, "T", identifies a "terminal" and will appear as a circle on the output screen. The letter "S" identifies a "station" and will appear as a rectangle on the screen. The letter "R" identifies a "repeater" and will be displayed as a truncated triangle. Input and output queues can be represented by placing an "I" or "O" as the first character of the place name. These nodes are displayed as small rectangles, large enough to contain a two digit number. Names may begin with letters other than those listed above. They will not, however, be displayed to the screen.

After the name is listed, on the same line, the screen location of the place is specified by means of an x, y coordinate system. The x and y values are in the range of 0-511, with the (0,0) point located at the top, left-hand corner of the graphics display unit.

The third item of information in the "places" line is a "plot on/off" entry. The user will frequently have places identified in the Petri-Net which are necessary control elements, but which do not need to be displayed on the output. A value of "1" will cause the place to be displayed with its label. A value of "0" is used for listing places which are not displayed to the output screen.

2. Transitions

After every place in the Petri-Net is listed (one line per place), the transitions are listed. As before, the first line specifies the total number of transitions to be read in.

Then, the transitions are each listed in a three-line format as follows:

```
Transition Name...X Coordinate...Y Coordinate...Plot Flag
Places into Transition
Places out of Transition
```

Transitions are named "TRXX", followed by their x, y coordinates, and a "1" or "0" to indicate whether they are to be displayed to the screen. The second line concerns the

input to the transition.

The first field of this line specifies the total number of inputs, and the following numbers indicate which places enter that transition. The numbers,

3 6 11 14

for instance, indicate that three places are inputs to this transition. The particular places are identified by their implicit, line-number ordering as entered in the list of places. In this example the three places are the 8th, 11th, and 14th places entered in the input file.

The third line of the transition entry concerns the outputs from that transition. The format is identical to that of the line above, i.e., the numbers

2 3 4

indicate that the transition fires to two outputs, the 3rd and 4th places listed in part one of the input file.

3. Initial Marking

After all the places and transitions are listed, the Petri-Net must be given its initial state of marking. The initial placement of tokens is specified by the following format:

MARK	Line # of Place	# of Tokens
------	-----------------	-------------

Thus, the entry "MARK 3 1" specifies that at the beginning of the simulation, Place 3 (the third entry in the list of places) is marked with one token. Several places may

be marked initially, but each marking requires a separate line.

Every line in the input file begins in the first column. Do not indent the beginning character of the line. The fields within each line must be separated by one (or more) blank spaces. The final line of the input file is the command "END".

4. Execution

When the user completes the input file he should exit from the edit mode and is now ready to execute the program "simulator" by typing "simulator.out". The program will ask him which input file he wishes to read, and the user responds by typing "RUNXX" (the file he previously created) and a <cr>.

C. THE OUTPUT FILES

The program "simulator" produces six separate output files. When RUN01 is entered into "simulator", files named RUN01A, RUN01B, RUN01X, RUN01Y, and RUN01Z are produced. The files suffixed with A through C are formatted files. Files X through Z are unformatted. Files A and X contain the essential data structures that have been read in "simulator" from the input file. Files B and Y contain the markings for each place at successive time frames which will appear on the graphics output. Files C and Z contain information concerning which links or transitions fire at any particular time frame and are used to highlight present activity on the screen. The graphics display programs, written in "C",

require unformatted input files. The formatted files are necessary for the user to validate that correct input data is reaching the graphics program, and to troubleshoot when locating a problem. Examples of these files are contained in Appendices C through E.

D. USER OPTIONS

1. Choice of Programs

After the program "simulator" has executed the input file, the "C" programs read the output files and display the results of the simulation to the screen. At this point the user has several options concerning the method of display.

There are two separate programs the user can select--"transgraph" and "linkgraph". By executing "transgraph" the viewer will be able to observe the nodes of the network together with their associated transitions. "Linkgraph" does not display the transitions, but links node-to-node in the common way that communications networks are most frequently represented. For simple networks or to explain the basic working of Petri-Nets, the user will probably desire to see the transitions. "Transgraph" will only run with less than 100 places. For more intricate networks that contain several hundred places and transitions, the "linknode" program is necessary to avoid congestion on the screen.

2. User Questions and Responses

After selecting which program to run, the user is given a series of questions from the CRT. Question one asks

the user to select which input file he wishes to execute. He responds with the command RUN X. For instance, if the input file named RUN21 has been executed by "simulator", and the user wishes to see the results, he enters RUN21X in response to question one. (Numerous files may be waiting in the users' directory which can be called in for display.)

Question 2 asks the user if he wants to view the data structures of the program. By entering a "1", the data structures will be printed to the CRT. A "0" will move on to the next question without viewing the data structures.

Question 3 asks the user which of three versions of the program he wishes to see. Version 1 displays the marking of tokens in the conventional Petri-Net fashion, with numbers printed at the center of the places. Version 2 represents tokens by single, yellow boxes printed inside the nodes. These boxes are designed to represent "packets" of information in the packet switching concept. As previously described, each node has the capacity to hold seven packets. If the number of packets goes over seven, a red overflow number appears beside the saturated place.

The third version is the multi-routing, multi-destination version. Version 3 uses color in a unique way. Because packets may be originating at different nodes and traveling to several destinations, the linking channels require two-way transmission. The graphics display is color-coded to highlight this information. Packets traveling to a particular place are colored to match the label of that

place. For instance, when a green box appears in the network, the viewer can trace its progress to the destination whose label is displayed in green. In Version 3 (multi-routing), a packet originating at a certain node and destined for another particular node may take different paths to arrive at the destination.

The user at this point must select one of the three versions by entering a "1", "2", or "3" followed by a <cr>.

Question 4 asks the user to select one of the three Genisco graphics terminals in the C3 lab on which the output will appear. Correct entries to this query are "0", "1", or "2".

The fifth question asks the user if he desires a time pause of two seconds duration between time frames of the simulation. If he desires the capability to look closely at each network snapshot, he enters a "1". If not, he should enter a "0" and the simulation will run without pauses. This gives the viewer more of a "real-time" impression.

At the end of this final question, the screen displays the initial condition of the network. After noting the initial condition, the user should type a carriage return to continue execution.

If the user has typed a "1" in response to question 5, he also has the ability to indefinitely suspend execution of the program at any time frame. He can interrupt the program by typing a "BRK" from the CRT keyboard. After studying that particular snapshot of network status, he may

continue normal execution by typing a carriage return.

This interrupt does not work if the program is executing without pauses. Typing two consecutive "PRK's" will enable the user to exit the program entirely.

3. The "Highlighting" Feature

The linking algorithm in both "transgraph" and "linkgraph" draws lines in a dark blue color. The highlighting feature in both programs changes the blue connecting link color to bright yellow on those links which are carrying traffic at any particular time frame. This feature performs in the following sequence. (1) The link lights up at the point where the future action will occur. (2) The packet(s) in question moves from one end of the highlighted link to the other. (3) The highlight remains on the link to emphasize where the action occurred. The user will notice that direction of movement on the highlighted link is indicated by an arrow pointing in the appropriate direction. See Figures 19 and 20.

E. UNIQUE INSTRUCTIONS FOR VERSION 3:

In order for the user to implement the capabilities of Version 3, special network design information must be included in the input file. This paragraph describes these special instructions.

The simulation is structured to function in a fixed routing manner. That is, the multi-routed paths are predefined by the user.

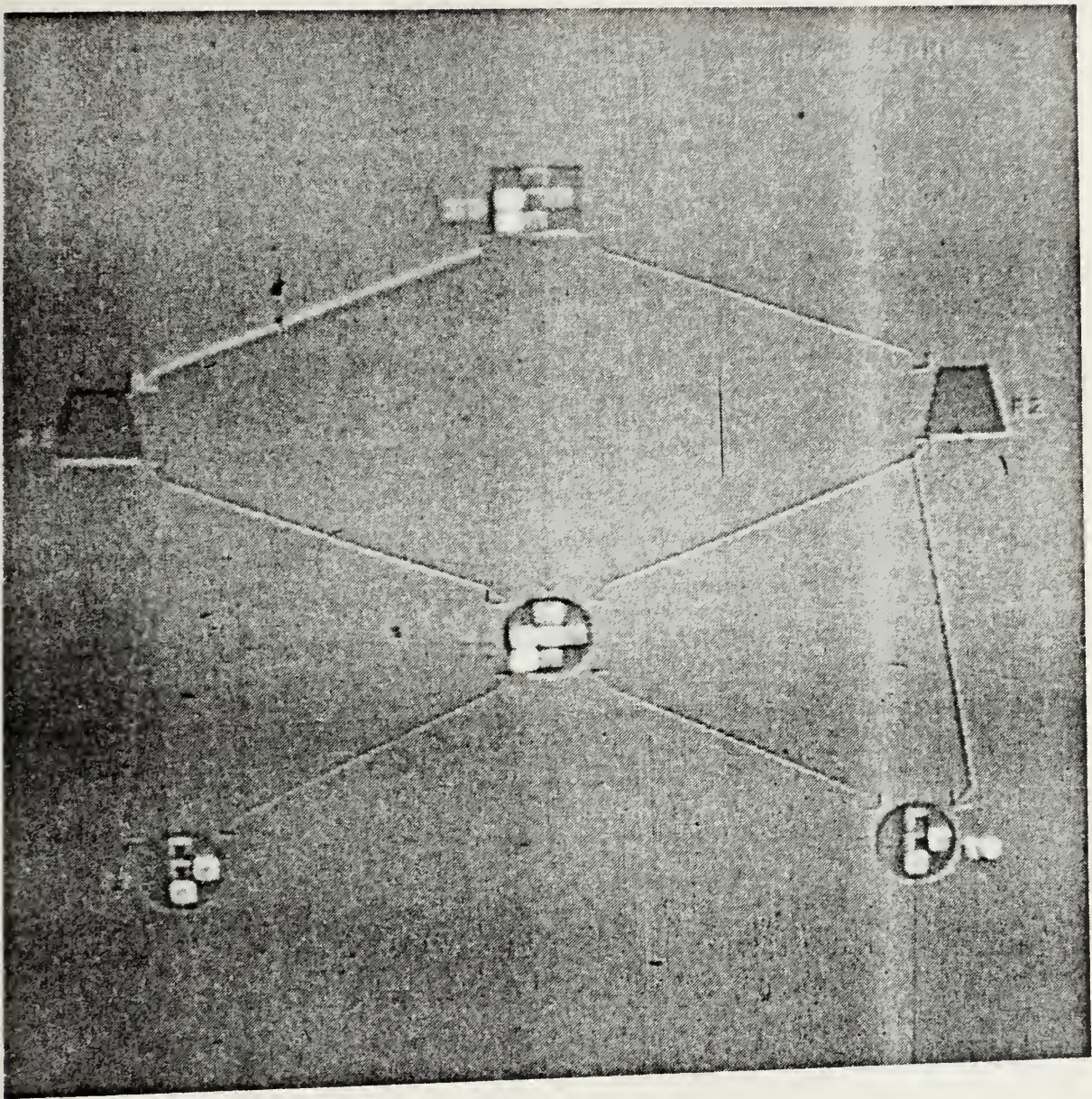


Figure 19.

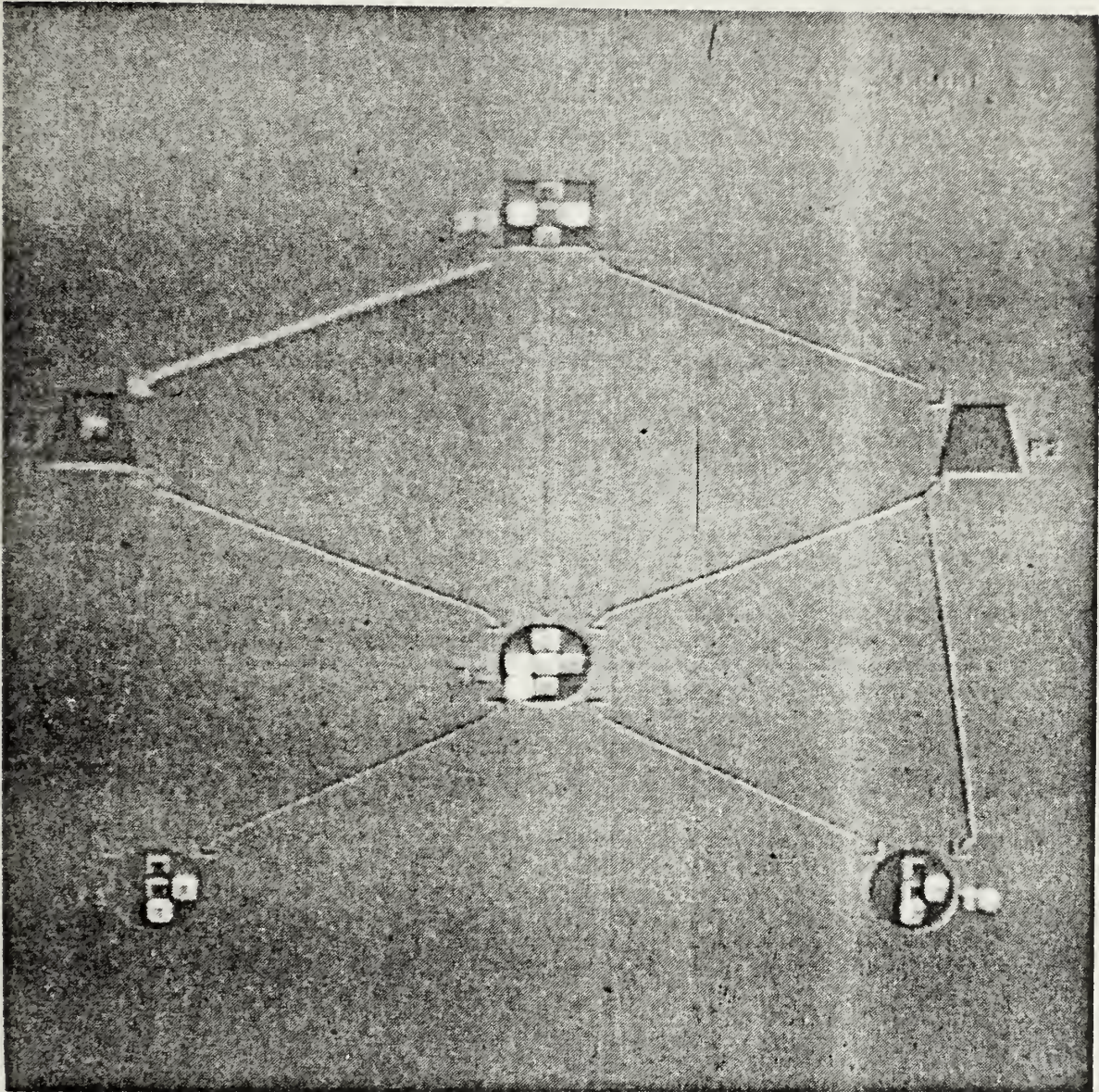


Figure 20.

There can be as many alternate paths from source to destination as the user wants to include. But once these paths are specified in the input file, they do not change dynamically during execution.

The multi-routing feature is made possible by "stacking" numerous places on top of one another and displaying these piggy-backed places at the same coordinates. In this manner the nodes appear on the screen as a single place, although in reality they may be buried several deep.

As the total network is conceptualized by the user, he must begin by mapping out all origins, all destinations and all relay nodes. Every node becomes unique to a particular path. For instance, a simple case would be to send a packet from T9 to T5 (see Figure 21.) by two different routes--Route 1 goes through R1 and Route 2 goes through R2.

In this case T9 and T5 would be "stacked" two deep. The routes are T9-R1-T5 and T9-R2-T5. Because they are plotted at the same point, the terminals appear as a single node.

The "header" information to perform this routing is contained in the name of the place. In Version 3, every node that will be displayed to the screen must be assigned a seven unit name in the input file.

The first unit of the name specifies the type of figure to be displayed ("T", "S", "R", "I", "O" as previously explained). The second unit specifies the color that the name will be printed in. This number is derived from the particular color table (16 colors are in a color table)

that is being used by the graphics program. The color of the label must correspond to the color of the packets bound for that route's destination. Units 3 and 4 of the name designate the route number that the node lies upon. Route numbers are arbitrarily given by the user and utilized for his own identification purposes. Unit 5 designates the color of the packet that will travel along that particular route. Every node on that route have the same color designator. The packet color of the route is determined by the destination of that route. Units 6 and 7 of the name specify how many nodes are stacked at that location. Places that are stacked must be listed together in the input file.

An example of this 7 unit name might appear as follows:

T205419

Field 1 designates that a circle will be drawn.

Field 2 specifies that the label will be displayed in color number 2 of the program's color table.

Fields 3 and 4 show the place on route number 05.

Field 5 ensures that every packet which passes through this place will be displayed in color number 4.

Fields 6 and 7 specify that places are stacked 19 deep at this coordinate.

As the program executes, the stacking algorithm totals all the packets which are located at a particular stacked location and displays that total number of boxes in the appropriate colors.

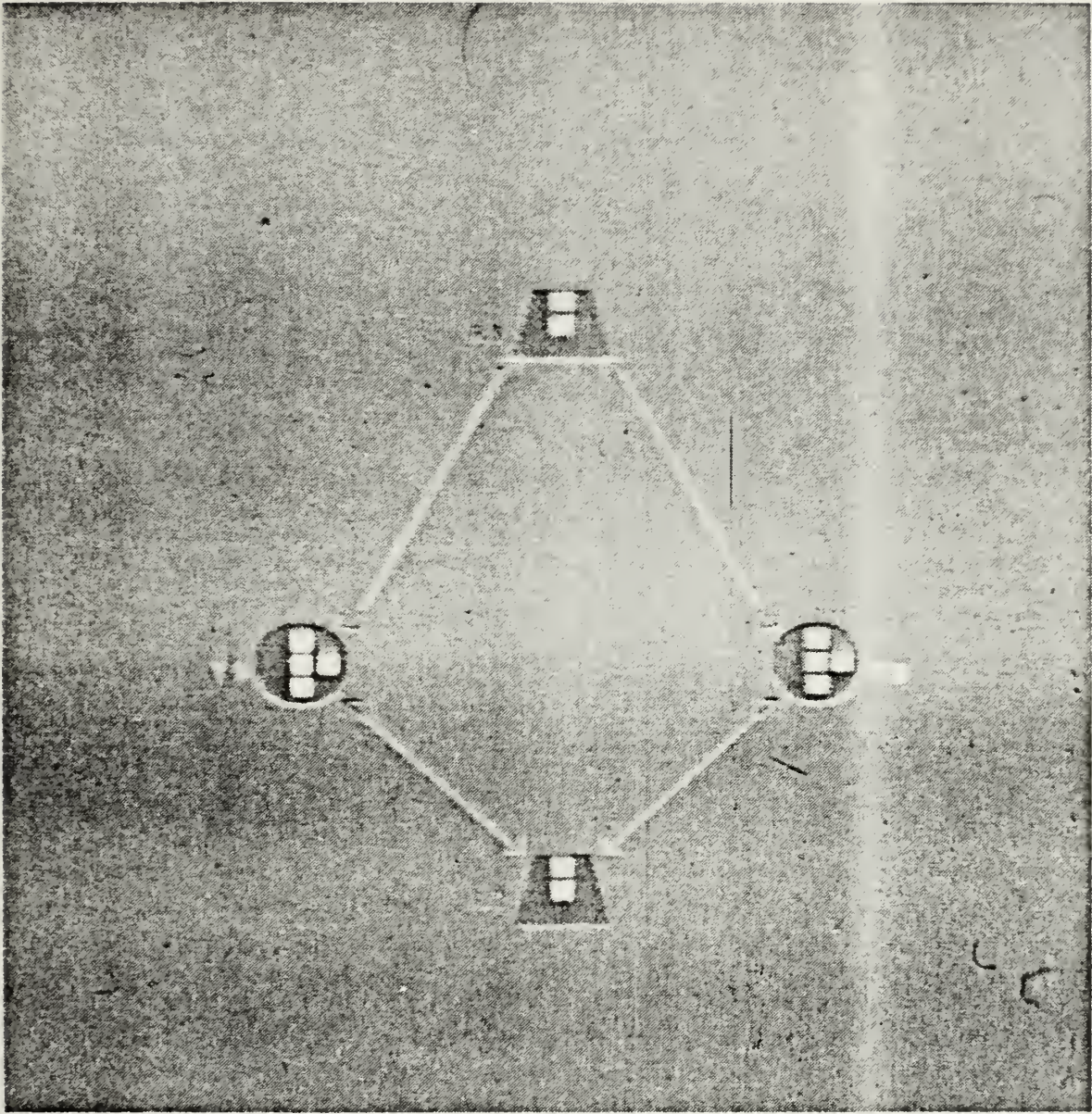


Figure 21.

RUN03 Page 1 Fri Feb 8 05:09:47 1980

```
1 10
2 T1 150 100 1
3 T2 150 250 1
4 T3 150 400 1
5 T4 350 250 1
6 G1 0 0 0
7 G2 0 0 0
8 G3 0 0 0
9 G4 0 0 0
10 G5 0 0 0
11 G6 0 0 0
12 8
13 TR1 250 175 1
14 2 1 2
15 1 4
16 TR2 250 325 1
17 2 2 3
18 2 4 8
19 TR3 0 0 0
20 1 5
21 2 1 6
22 TR4 0 0 0
23 1 6
24 1 10
25 TR5 0 0 0
26 1 7
27 2 2 9
28 TR6 0 0 0
29 1 8
30 1 3
31 TR7 0 0 0
32 1 9
33 1 7
34 TR8 0 0 0
35 1 10
36 1 5
37 MARK 5 1
38 MARK 7 1
39 MARK 3 1
40 END
```



```

1  NEXTREV= 11
2      1      0      150      100      1      2
3      5      0      150      250      1      2
4      5      1      150      400      1      2
5      7      0      350      250      1      2
6      9      1      0      0      0      2
7      11     0      0      0      0      2
8      13     1      0      0      0      2
9      15     0      0      0      0      2
10     17     0      0      0      0      2
11     19     0      0      0      0      2
12

```

```

13  NEXTIRN= 9
14     21      1      4      250      175      1      3
15     24      6      2      250      325      1      3
16     27     12     14      0      0      0      3
17     30     17     10      0      0      0      3
18     33     21     23      0      0      0      3
19     36     25     24      0      0      0      3
20     39     30     32      0      0      0      3
21     42     34     35      0      0      0      3
22

```

```

23  IOTABLE:
24     2  1  2  1  4  2  2  3  2  4
25     8  1  5  2  1  6  1  6  1 10
26     1  7  2  2  9  1  8  1  3  1
27     9  1  7  1 10  1  5  0
28

```

```

29  NAMES:  NEXTNAME= 45
30  T1T2T3T4G1G2G3G4G5G6I1I2I3I4I5I6I7I8
31

```


RUN03B Page 1 Fri Feb 8 05:19:47 1980

1	1	1	1	0	0	1	0	0	1	0
2	0	0	1	1	0	0	1	0	0	1
3	0	1	1	1	1	0	0	0	1	0
4	1	0	0	2	0	1	1	1	0	0
5	1	1	1	2	0	0	0	0	1	1
6	0	0	1	3	1	0	1	0	0	0
7	1	1	1	3	0	1	0	0	1	0
8	0	0	1	4	0	0	1	0	0	1
9	0	1	1	4	1	0	0	0	1	0
10	1	0	0	5	0	1	1	1	0	0
11	1	1	1	5	0	0	0	0	1	1
12	0	0	1	6	1	0	1	0	0	0
13	1	1	1	6	0	1	0	0	1	0
14	0	0	1	7	0	0	1	0	0	1
15	0	1	1	7	1	0	0	0	1	0
16	1	0	0	8	0	1	1	1	0	0
17	1	1	1	8	0	0	0	0	1	1
18	0	0	1	9	1	0	1	0	0	0
19	1	1	1	9	0	1	0	0	1	0
20	0	0	1	10	0	0	1	0	0	1
21	0	1	1	10	1	0	0	0	1	0
22	1	0	0	11	0	1	1	1	0	0
23	1	1	1	11	0	0	0	0	1	1
24	0	0	1	12	1	0	1	0	0	0
25	1	1	1	12	0	1	0	0	1	0
26	0	0	1	13	0	0	1	0	0	1
27	0	1	1	13	1	0	0	0	1	0
28	1	0	0	14	0	1	1	1	0	0
29	1	1	1	14	0	0	0	0	1	1
30	0	0	1	15	1	0	1	0	0	0
31	1	1	1	15	0	1	0	0	1	0
32	0	0	1	16	0	0	1	0	0	1
33	0	1	1	16	1	0	0	0	1	0
34	1	0	0	17	0	1	1	1	0	0
35	1	1	1	17	0	0	0	0	1	1
36	0	0	1	18	1	0	1	0	0	0
37	1	1	1	18	0	1	0	0	1	0
38	0	0	1	19	0	0	1	0	0	1
39	0	1	1	19	1	0	0	0	1	0
40	1	0	0	20	0	1	1	1	0	0
41	1	1	1	20	0	0	0	0	1	1
42	0	0	1	21	1	0	1	0	0	0
43	1	1	1	21	0	1	0	0	1	0
44	0	0	1	22	0	0	1	0	0	1
45	0	1	1	22	1	0	0	0	1	0
46	1	0	0	23	0	1	1	1	0	0
47	1	1	1	23	0	0	0	0	1	1
48	0	0	1	24	1	0	1	0	0	0
49	1	1	1	24	0	1	0	0	1	0
50	0	0	1	25	0	0	1	0	0	1

1 2
2 3 5
3 3
4 1 4 7
5 2
6 5 8
7 3
8 2 3 7
9 3
10 4 5 6
11 3
12 1 7 8
13 2
14 3 5
15 3
16 1 4 7
17 2
18 5 8
19 3
20 2 3 7
21 3
22 4 5 6
23 3
24 1 7 8
25 2
26 3 5
27 3
28 1 4 7
29 2
30 5 8
31 3
32 2 3 7
33 3
34 4 5 6
35 3
36 1 7 8
37 2
38 3 5
39 3
40 1 4 7
41 2
42 5 8
43 3
44 2 3 7
45 3
46 4 5 6
47 3
48 1 7 8
49 2
50 3 5
51 3
52 1 4 7
53 2
54 5 8
55 3
56 2 3 7
57 3
58 4 5 6
59 3
60 1 7 8

61 2
62 3 5
63 3
64 1 4 7
65 2
66 5 8
67 3
68 2 3 7
69 3
70 4 5 6
71 3
72 1 7 8
73 2
74 3 5
75 3
76 1 4 7
77 2
78 5 8
79 3
80 2 3 7
81 3
82 4 5 6
83 3
84 1 7 8
85 2
86 3 5
87 3
88 1 4 7
89 2
90 5 8
91 3
92 2 3 7
93 3
94 4 5 6
95 3
96 1 7 8
97 2
98 3 5
99 3
100 1 4 7

1	122			
2	S301212	250	100	1
3	S302212	250	100	1
4	S303112	250	100	1
5	S304112	250	100	1
6	S305012	250	100	1
7	S306012	250	100	1
8	S307312	250	100	1
9	S308312	250	100	1
10	S315312	250	100	1
11	S316312	250	100	1
12	S319312	250	100	1
13	S320312	250	100	1
14	R102205	75	200	1
15	R103105	75	200	1
16	P110105	75	200	1
17	R114205	75	200	1
18	R115305	75	200	1
19	R205005	425	200	1
20	R208305	425	200	1
21	R212005	425	200	1
22	R218205	425	200	1
23	R220305	425	200	1
24	T201216	250	300	1
25	T202216	250	300	1
26	T204116	250	300	1
27	T206016	250	300	1
28	T207316	250	300	1
29	T208316	250	300	1
30	T209116	250	300	1
31	T210116	250	300	1
32	T211016	250	300	1
33	T212016	250	300	1
34	T213216	250	300	1
35	T214216	250	300	1
36	T216316	250	300	1
37	T217216	250	300	1
38	T218216	250	300	1
39	T219316	250	300	1
40	T103108	100	400	1
41	T104108	100	400	1
42	T109108	100	400	1
43	T110108	100	400	1
44	T113208	100	400	1
45	T114208	100	400	1
46	T115308	100	400	1
47	T116308	100	400	1
48	T005008	400	400	1
49	T006008	400	400	1
50	T011008	400	400	1
51	T012008	400	400	1
52	T017208	400	400	1
53	T018208	400	400	1
54	T019308	400	400	1
55	T020308	400	400	1
56	B1	100	100	0
57	B2	100	100	0
58	B3	100	100	0
59	B4	100	100	0
60	B5	100	100	0

61 B6	100	100	0
62 B7	100	100	0
63 B8	100	100	0
64 B9	100	100	0
65 B10	100	100	0
66 B11	100	100	0
67 B12	100	100	0
68 B13	100	100	0
69 B14	100	100	0
70 B15	100	100	0
71 B16	100	100	0
72 B17	100	100	0
73 B18	100	100	0
74 B19	100	100	0
75 B20	100	100	0
76 B21	100	100	0
77 B22	100	100	0
78 B23	100	100	0
79 B24	100	100	0
80 B25	100	100	0
81 B26	100	100	0
82 B27	100	100	0
83 B28	100	100	0
84 B29	100	100	0
85 B30	100	100	0
86 B31	100	100	0
87 B32	100	100	0
88 B33	100	100	0
89 B34	100	100	0
90 A1	500	500	0
91 A2	500	500	0
92 A3	500	500	0
93 A4	500	500	0
94 A5	500	500	0
95 A6	500	500	0
96 A7	500	500	0
97 A8	500	500	0
98 A9	500	500	0
99 A10	500	500	0
100 A11	500	500	0
101 A12	500	500	0
102 A13	500	500	0
103 A14	500	500	0
104 A15	500	500	0
105 A16	500	500	0
106 A17	500	500	0
107 A18	500	500	0
108 A19	500	500	0
109 A20	500	500	0
110 A21	500	500	0
111 A22	500	500	0
112 A23	500	500	0
113 A24	500	500	0
114 A25	500	500	0
115 A26	500	500	0
116 A27	500	500	0
117 A28	500	500	0
118 A29	500	500	0
119 A30	500	500	0
120 A31	500	500	0

121	A32	500	500	0
122	A33	500	500	0
123	A34	500	500	0
124	68			
125	TR1	220	200	1
126	2 1 56			
127	1 23			
128	TR2	140	140	1
129	2 2 58			
130	1 13			
131	TR3	140	260	1
132	2 13 60			
133	1 24			
134	TR4	100	100	0
135	2 3 62			
136	1 14			
137	TR5	100	100	0
138	2 14 64			
139	1 39			
140	TR6	100	100	0
141	2 4 66			
142	1 25			
143	TR7	100	100	0
144	2 25 68			
145	1 40			
146	TR8	330	160	1
147	2 5 70			
148	1 18			
149	TR9	375	320	1
150	2 18 72			
151	1 47			
152	TR10	100	100	0
153	2 6 74			
154	1 26			
155	TR11	100	100	0
156	2 26 76			
157	1 48			
158	TR12	280	200	1
159	2 27 78			
160	1 7			
161	TR13	365	260	1
162	2 28 80			
163	1 19			
164	TR14	100	100	0
165	2 19 82			
166	1 8			
167	TR15	175	400	1
168	2 29 84			
169	1 41			
170	TR16	170	230	1
171	2 30 86			
172	1 15			
173	TR17	50	300	1
174	2 15 88			
175	1 42			
176	TR18	450	300	1
177	2 54 92			
178	1 22			
179	TR19	360	140	1
180	2 22 90			

181	1	12			
182	TR20		100	100	0
183	2	53	96		
184	1	38			
185	TR21		100	100	0
186	2	38	94		
187	1	11			
188	TR22		100	100	0
189	2	52	100		
190	1	21			
191	TR23		330	230	1
192	2	21	98		
193	1	37			
194	TR24		325	335	1
195	2	51	102		
196	1	36			
197	TR25		100	100	0
198	2	46	106		
199	1	35			
200	TR26		100	100	0
201	2	35	104		
202	1	10			
203	TR27		100	350	1
204	2	45	108		
205	1	17			
206	TR28		170	160	1
207	2	17	110		
208	1	9			
209	TR29		100	100	0
210	2	44	112		
211	1	16			
212	TR30		100	100	0
213	2	16	114		
214	1	34			
215	TR31		180	310	1
216	2	43	116		
217	1	33			
218	TR32		100	100	0
219	2	32	120		
220	1	20			
221	TR33		100	100	0
222	2	20	118		
223	1	50			
224	TR34		300	400	1
225	2	31	122		
226	1	49			
227	TR35		100	100	0
228	1	55			
229	2	57	56		
230	TR36		100	100	0
231	1	57			
232	2	59	58		
233	TR37		100	100	0
234	1	59			
235	2	60	61		
236	TR38		100	100	0
237	1	61			
238	2	63	62		
239	TR39		100	100	0
240	1	63			

241	2	65	64			
242	TR40			100	100	0
243	1	65				
244	2	67	66			
245	TR41			100	100	0
246	1	67				
247	2	69	68			
248	TR42			100	100	0
249	1	69				
250	2	71	70			
251	TR43			100	100	0
252	1	71				
253	2	73	72			
254	TR44			100	100	0
255	1	73				
256	2	75	74			
257	TR45			100	100	0
258	1	75				
259	2	77	76			
260	TR46			100	100	0
261	1	77				
262	2	79	78			
263	TR47			100	100	0
264	1	79				
265	2	81	80			
266	TR48			100	100	0
267	1	81				
268	2	83	82			
269	TR49			100	100	0
270	1	83				
271	2	85	84			
272	TR50			100	100	0
273	1	85				
274	2	87	86			
275	TR51			100	100	0
276	1	87				
277	2	89	88			
278	TR52			100	100	0
279	1	89				
280	2	91	92			
281	TR53			100	100	0
282	1	91				
283	2	93	90			
284	TR54			100	100	0
285	1	93				
286	2	95	96			
287	TR55			100	100	0
288	1	95				
289	2	97	94			
290	TR56			100	100	0
291	1	97				
292	2	99	100			
293	TR57			100	100	0
294	1	99				
295	2	101	98			
296	TR58			100	100	0
297	1	101				
298	2	103	102			
299	TR59			100	100	0
300	1	103				

301 2 105 106
 302 TR60 100 100 0
 303 1 105
 304 2 107 104
 305 TR61 100 100 0
 306 1 107
 307 2 109 108
 308 TR62 100 100 0
 309 1 109
 310 2 111 110
 311 TR63 100 100 0
 312 1 111
 313 2 113 112
 314 TR64 100 100 0
 315 1 113
 316 2 115 114
 317 TR65 100 100 0
 318 1 115
 319 2 117 116
 320 TR66 100 100 0
 321 1 117
 322 2 119 120
 323 TR67 100 100 0
 324 1 119
 325 2 121 118
 326 TR68 100 100 0
 327 1 121
 328 2 55 122
 329 MARK 1 1
 330 MARK 2 1
 331 MARK 3 1
 332 MARK 4 1
 333 MARK 5 1
 334 MARK 6 1
 335 MARK 27 1
 336 MARK 28 1
 337 MARK 29 1
 338 MARK 30 1
 339 MARK 31 1
 340 MARK 32 1
 341 MARK 43 1
 342 MARK 44 1
 343 MARK 45 1
 344 MARK 46 1
 345 MARK 51 1
 346 MARK 52 1
 347 MARK 53 1
 348 MARK 54 1
 349 MARK 61 1
 350 END

1				
2				
3		THESIS-RELATED PROGRAMS & TEXT		
4				
5				
6				
7	FILE	DESCRIPTION	VERSION	EXECUTION
8				
9				
10	DIRECTORY	LIST OF THESIS FILES	-	aa DIRECTORY
11	RUN01X	GENERATOR	2	transgraph
12	RUN02X	ENABLE TRANSITION	2	transgraph
13	RUN03X	DYNAMIC CONFLICTS	2	transgraph
14	RUN04X	HIERARCHICAL STRUCTURE	2	linkgraph
15	RUN05X	HIERARCHICAL STRUCTURE	2	transgraph
16	RUN06X	RING STRUCTURE	2	linkgraph
17	RUN07X	FEEDBACK GENERATOR	1	transgraph
18	RUN09X	DEPICT SYNCHRONIZATION	1	transgraph
19	RUN11X	122 MULTIPLE INPUTS (5)	2	transgraph
20	RUN12X	ALL-TO-ALL INPUT	2	transgraph
21	RUN13X	CONSUMER-PRODUCER	2	transgraph
22	RUN14X	CONSUMER-PRODUCER	2	transgraph
23	RUN15X	INITIAL RUN RUN09X	2	linkgraph
24	RUN16X	SYNCHRONIZATION II	2	transgraph
25	RUN20X	PACKET-RADIO NETWORK	3	linkgraph
26	RUN21X	SIMPLE MULTI-ROUTING	3	link/transgraph
27	RUN22X	CONCURRENT NETWORK	3	linkgraph
28	RUN23X	TIME-SLOTTED COMMUNICATIONS	3	linkgraph
29	RUN24X	PRIORITY TIME-SLOTTED	3	linkgraph
30	RUN25X	CURRENT NETWORK (-)	3	linkgraph
31	RUN26X	TIME-SLOTTED COMMUNICATION(-)	3	linkgraph
32	RUN27X	PRIORITY TIME-SLOTTED (-)	3	linkgraph
33	RUN01-29	INPUT FILES TO SIMULATOR	-	simulator.out
34	advisors.c	THESIS ADVISORS	-	advisors
35	end.c	END FRAME FOR FILM	-	end
36	experiment.c	INTRODUCE RESULTS OF EXPERIMENT	-	experiment
37	intro.c	INTRODUCTION FOR FILM	-	intro
38	linear.c	CODE-FILE SIZE RELATIONSHIP	-	linear
39	linkgraph.c	COMMON-LINK GRAPHICS	-	linkgraph
40	ops.c	TITLE FRAME FOR OPS	-	ops
41	outline.c	SCREENING MODELS	-	outline
42	simulator	PERFORM SIMULATOR	-	simulator.out
43	title.c	TITLE FRAME FOR FILM	-	title
44	transgraph.c	TRANSITION GRAPHICS	-	transgraph


```

1 PROGRAM SIMULATOR
2
3 C ORIGINAL VERSION OF THIS PROGRAM (PROGRAM TESTINT) WRITTEN BY L.A.COXS
4 C MODIFIED TO OPERATE ON UNIX (PDP 11/70) BY S.C.JENNINGS & R.J.HARTELO
5 C PROGRAM SIMULATOR READS USER INPUT FILE AND PRODUCES 6 OUTPUT FILES
6
7 C MAINLINE
8
9 CALL INIT
10 CALL INPUT1
11 CALL DUMPPP
12 CALL MOVENET(150)
13 CALL EXIT
14 END
15
16
17 SUBROUTINE INIT
18
19 C INIT OPENS USER INPUT FILE & CREATES 6 OUTPUT FILES
20 C FNAME1 STORES THE INPUT FILE ----- RUN..
21 C FNAME2 STORES THE FORMATTED INPUT DATA STRUCTURES ----- RUN..A
22 C FNAME3 STORES THE FORMATTED ITERATIONS OF THE NETWORK - RUN..B
23 C FNAME4 STORES THE FORMATTED LINKS OF THE NETWORK ----- RUN..C
24 C FNAME5 STORES THE UNFORMATTED GRAPHICS INPUT ----- RUN..X
25 C FNAME6 STORES THE UNFORMATTED GRAPHICS ITERATIONS ----- RUN..Y
26 C FNAME7 STORES THE UNFORMATTED GRAPHICS STATES ----- RUN..Z
27
28
29 BYTE FNAME1
30 BYTE FNAME2
31 BYTE FNAME3
32 BYTE FNAME4
33 BYTE FNAME5
34 BYTE FNAME6
35 BYTE FNAME7
36
37 COMMON/USRFILE/FNAME1(6),FNAME2(7),FNAME3(7),FNAME4(7),FNAME5(7),FNAME6(7)
38 1 FNAME7(7)
39 COMMON/EVENT/IEVENT(400,6),NXTTIT
40 COMMON/ITRANS/ITRANS(400,7),NXTITRN,IINTR,IISTORE(100)
41 BYTE NAMES
42 COMMON/NAME/NAMES(4000),NXTNAM
43 COMMON/IOTAB/IOTAB(4000),NXTITRE
44 1000 FORMAT(' INITIALIZING PROGRAM')
45 TYPE 1000
46 1002 FORMAT(' BEGIN TEST-GRAPH-NET')
47 TYPE 1002
48 2007 TYPE 2000
49 2000 FORMAT (' ***-> INPUT FILE? NAME MUST BE ENTERED AS: RUN01 - RUN90
50 ACCEPT 2001, FNAME1
51 2001 FORMAT (6A1)
52 FNAME1(6) = 0
53 OPEN (UNIT = 1, NAME = FNAME1, TYPE = 'OLD', ERR = 2006)
54 GO TO 2004
55 2006 TYPE 2002, FNAME1
56 2002 FORMAT (' ERROR OPENING FILE ',X6A1)
57 GO TO 2007
58 2004 DO 2005 I = 1, 5
59 FNAME2(I) = FNAME1(I)
60 FNAME3(I) = FNAME1(I)

```



```

61      FNAM4(I) = FNAM1(I)
62      FNAM5(I) = FNAM1(I)
63      FNAM6(I) = FNAM1(I)
64      FNAM7(I) = FNAM1(I)
65  2005 CONTINUE
66      FNAM2(6) = 'A'
67      FNAM2(7) = 0
68      FNAM3(6) = 'B'
69      FNAM3(7) = 0
70      FNAM4(6) = 'C'
71      FNAM4(7) = 0
72      FNAM5(6) = 'X'
73      FNAM5(7) = 0
74      FNAM6(6) = 'Y'
75      FNAM6(7) = 0
76      FNAM7(6) = 'Z'
77      FNAM7(7) = 0
78      NXTEVT=1
79      NXTTRN=1
80      NXTNAM=1
81      NXTTRE=1
82  3000 FORMAT(' INITIALIZATION COMPLETE')
83      TYPE 3000
84      RETURN
85      END
86
87
88      SUBROUTINE DUMPPP
89
90  C      OPENS FNAM2 & FNAM4
91      BYTE      FNAM1
92      BYTE      FNAM2
93      BYTE      FNAM3
94      BYTE      FNAM4
95      BYTE      FNAM5
96      BYTE      FNAM6
97      BYTE      FNAM7
98      COMMON/USRFIL/FNAM1(6),FNAM2(7),FNAM3(7),FNAM4(7),FNAM5(7),FNAM6(7):
99      1          FNAM7(7)
100     COMMON/EVENT/IEVENT(400,6),NXTEVT
101     COMMON/TRANS/ITRANS(400,7),NXTTRN,IINTR,IISTORE(100)
102     COMMON/IOTAB/IOTABL(4000),NXTTRE
103     BYTE NAMES
104     COMMON/NAME/NAMES(4000),NXTNAM
105     OPEN(UNIT=1,NAME=FNAM2,TYPE='NEW',INITIALSIZE=40000)
106  1000 FORMAT(' NXTEVT=',I4)
107  1001 FORMAT(5x,6I8)
108      WRITE(1,1000) NXTEVT
109      DO 1500 I=1,NXTEVT-1
110  1500 WRITE(1,1001) (IEVENT(I,J),J=1,6)
111  2000 FORMAT(/,' NXTTRN=',I4)
112  2001 FORMAT(1x,7I8)
113      WRITE(1,2000) NXTTRN
114      DO 2500 I=1,NXTTRN-1
115  2500 WRITE(1,2001) (ITRANS(I,J),J=1,7)
116      2500 CONTINUE
117
118  3000 FORMAT(/,' IOTABLE:',/,60(10I4,/))
119      WRITE(1,3000) (IOTABL(I),I=1,NXTTRE)
120  4000 FORMAT(/,' NAMES: NXTNAM=',I4)

```



```

121 4001 FORMAT(X,100A1)
122 WRITE(1,4000) NXTNAM
123 WRITE(1,4001) (NAMES(I),I=1,NXTNAM)
124 5000 FORMAT(1H1)
125 WRITE(1,5000)
126 CLOSE(UNIT=1,DISPOSE='SAVE')
127
128 OPEN(UNIT=1,NAME=FNAMS,TYPE='NEW',INITIALSIZE=14000,
129 1 FORM='UNFORMATTED')
130 WRITE(1) (NXTEVT)
131 DO 5001 I=1,NXTEVT-1
132 5001 WRITE(1) (IEVENT(I,J),J=1,6)
133 WRITE(1) (NXTTRN)
134 DO 5002 I=1,NXTTRN-1
135 5002 WRITE(1) (ITRANS(I,J),J=1,7)
136 WRITE(1) (NXTTIRE)
137 WRITE(1) (IOTABL(I),I=1,NXTTIRE)
138 WRITE(1) (NXTNAM)
139 WRITE(1) (NAMES(I),I=1,NXTNAM)
140 CLOSE(UNIT=1,DISPOSE='SAVE')
141 RETURN
142 END
143
144
145 SUBROUTINE INPUT1
146
147 COMMON/SCAN/IWORD(15,10),NUMBER
148 8000 FORMAT(' INPUT BEGINS')
149 8001 FORMAT(' INPUT COMPLETE')
150 TYPE 8000
151 I=0
152 CALL SCANR
153 CALL XINTGP(1,I)
154 DO 1000 J=1,I
155 CALL INPUTE
156 1000 CONTINUE
157
158 CALL INPUTT
159
160 2000 CONTINUE
161 CALL SCANR
162 IF(MATCHS(1,'END',3).EQ.1) GO TO 3000
163 IF(MATCHS(1,'MARK',4).EQ.1) CALL MARKER
164 GO TO 2000
165
166 3000 TYPE 8001
167 CLOSE(UNIT=1,DISPOSE='SAVE')
168 RETURN
169 END
170
171
172 SUBROUTINE INPUTE
173
174 COMMON/EVENT/IEVENT(400,6),NXTEVT
175 BYTE IWORD
176 COMMON/SCAN/IWORD(15,10),NUMB
177 J=0
178 C READ A SINGLE EVENT LINE FROM INPUT AND
179 C STORE IT APPROPRIATELY
180 CALL SCANR

```



```

181      CALL STONAM(1,IEVENT(NXTEVT,1),IEVENT(NXTEVT,6))
182
183      DO 1000 I=2,4
184      CALL XINTGR(I,J)
185      1000 IEVENT(NXTEVT,I+1)=J
186
187      NXTEVT=NXTEVT+1
188      IF(NXTEVT.GT.400) GO TO 9000
189      RETURN
190      9000 TYPE 9900
191      9900 FORMAT(' EVENT/PLACE TABLE OVERFLOW')
192      CALL EXIT
193      RETURN
194      END
195
196
197      SUBROUTINE INPUTT
198
199      COMMON/TRANS/ITRANS(400,7),NXTTRN,IINTR,IISTORE(100)
200      BYTE IWORD
201      COMMON/SCAN/IWORD(15,10),NUMB
202
203      I=0
204      K=0
205      CALL SCANR
206      CALL XINTGR(1,I)
207
208      DO 1000 J=1,I
209      CALL SCANR
210      CALL STONAM(1,ITRANS(NXTTRN,1),ITRANS(NXTTRN,7))
211
212      DO 2000 L=1,3
213      LL=L+3
214      CALL XINTGR(L+1,K)
215      2000 ITRANS(NXTTRN,LL)=K
216
217      CALL SCANR
218      CALL STOTOT(NUMB,ITRANS(NXTTRN,2))
219      CALL SCANR
220      CALL STOTOT(NUMB,ITRANS(NXTTRN,3))
221      NXTTRN=NXTTRN+1
222      IF(NXTTRN.GT.400) GO TO 9000
223      1000 CONTINUE
224
225      RETURN
226
227      9000 TYPE 9900
228      9900 FORMAT(' TRANSITION TABLE OVERFLOW')
229      CALL EXIT
230      RETURN
231      END
232
233
234      SUBROUTINE STONAM(NWORD,NPOINT,KOUNT)
235
236      C      STORE STRING 'NWORD' FROM SCANNER INTO
237      C      NAME TABLE AND RETURN A POINTER 'NPOINT'
238
239      BYTE IWORD,NAMES,BLANK
240      COMMON/SCAN/IWORD(15,10),NUMB

```



```

241      COMMON/NAME/NAMES(4000),NXTNAM
242      DATA BLANK/1H /
243      DO 1000 I=1,10
244      KOUNT=I-1
245      IF(IWORD(NWORD,I).EQ.BLANK) GO TO 2000
246 1000  CONTINUE
247 2000  CONTINUE
248
249      IF(NXTNAM+KOUNT .GT.4000) GO TO 9000
250
251      DO 3000 I=1,KOUNT
252 3000  NAMES(NXTNAM+I-1)=IWORD(NWORD,I)
253
254      NPOINT=NXTNAM
255      NXTNAM=NXTNAM+KOUNT
256      RETURN
257
258 9000  TYPE 9900
259 9900  FORMAT(' NAME TABLE OVERFLOW')
260      CALL EXIT
261      RETURN
262      END
263
264
265      SUBROUTINE STOLOT(NUMBER,LINK)
266
267      COMMON/IOTAB/IOTABL(4000),NXTTRE
268
269 C      STORE INPUTS AND OUTPUTS OF TRANSITIONS
270 C      IN THE TABLE, RETURN THE LINK
271
272      IF(NXTTRE+NUMBER .GT. 4000) GO TO 9000
273
274      K=0
275      DO 1000 I=1,NUMBER
276      CALL XINTGR(I,K)
277      J=(NXTTRE-1)+I
278 1000  IOTABL(J)=K
279
280      LINK=NXTTRE
281      NXTTRE=NXTTRE+NUMBER
282      RETURN
283
284 9000  TYPE 9900
285 9900  FORMAT(' IO TABLE OVERFLOW (TRANSITIONS)')
286      CALL EXIT
287      RETURN
288      END
289
290
291      SUBROUTINE MARKER
292
293      COMMON/EVENT/IEVENT(400,6),NXTEVT
294      I=0
295      J=0
296      CALL XINTGR(2,I)
297      CALL XINTGR(3,J)
298      IF(I.LT.1 .OR. I.GT.NXTEVT) RETURN
299      IEVENT(I,2)=J
300      RETURN

```



```

301      END
302
303
304
305      SUBROUTINE MOVENET(NTIMES)
306
307 C      EXECUTE THE PETRI-NET 'NTIMES' OR STEPS
308
309      COMMON/TRANS/ITRANS(400,7),NXTTRN,IINTR,IISTORE(100)
310      COMMON/EVENT/IEVENT(400,6),NXTEVT
311      BYTE      FNAM1
312      BYTE      FNAM2
313      BYTE      FNAM3
314      BYTE      FNAM4
315      BYTE      FNAM5
316      BYTE      FNAM6
317      BYTE      FNAM7
318      COMMON/USRFILE/FNAM1(6),FNAM2(7),FNAM3(7),FNAM4(7),FNAM5(7),FNAM6(7),
319 1      FNAM7(7)
320      DATA ITIME/1/
321
322 1000 FORMAT(' EXECUTING  TIME=',I4)
323
324      OPEN(UNIT=1,NAME=FNAM3,TYPE='NEW',INITIALSIZE=120000)
325      OPEN(UNIT=2,NAME=FNAM6,TYPE='NEW',FORM='UNFORMATTED',
326 1      INITIALSIZE=120000)
327      OPEN(UNIT=3,NAME=FNAM4,TYPE='NEW',INITIALSIZE=12000)
328      OPEN(UNIT=4,NAME=FNAM7,TYPE='NEW',FORM='UNFORMATTED',
329 1      INITIALSIZE=12000)
330
331
332      DO 2000 I=1,NTIMES
333      TYPE 1000,ITIME
334      IINTR=0
335      CALL MOVE
336 106 FORMAT(35I2)
337      WRITE(1,106) (IEVENT(J,2),J=1,NXTEVT-1)
338      WRITE(2) (IEVENT(J,2),J=1,NXTEVT-1)
339 107 FORMAT(I3)
340      WRITE(3,107) IINTR
341 108 FORMAT(100I3)
342      WRITE(3,108) (IISTORE(J), J=1,IINTR)
343      WRITE(4) IINTR
344      WRITE(4) (IISTORE(J),J=1,IINTR)
345
346      ITIME=ITIME+1
347 2000 CONTINUE
348
349      CLOSE(UNIT=4,DISPOSE='SAVE')
350      CLOSE(UNIT=3,DISPOSE='SAVE')
351      CLOSE(UNIT=2,DISPOSE='SAVE')
352      CLOSE(UNIT=1,DISPOSE='SAVE')
353
354      RETURN
355      END
356
357
358      SUBROUTINE MOVE
359
360 C      EXECUTE THE NET ONE STEP

```



```

361
362      COMMON/TRANS/ITRANS(400,7),NXTTRN,IINTR,IISTORE(100)
363      DIMENSION MARKS(400)
364      ITEST=0
365
366 C      CHECK ALL TRANSITIONS TO SEE WHICH ARE ENABLED
367      DO 0500 I=1,NXTTRN-1
368 0500 MARKS(I)=NABLED(I)
369      DO 1000 I=1,NXTTRN-1
370      IF(MARKS(I).EQ.0) GO TO 1000
371 0600 FORMAT(' DYNAMIC CONFLICT, TR#=',I4)
372
373      IF(NABLED(I).EQ.1) GO TO 0800
374      TYPE 0600,I
375      GO TO 1000
376 0800 CONTINUE
377      CALL UNMARK(I,ITEST)
378      IF(ITEST.EQ.1) GO TO 1000
379      IINTR=IINTR+1
380      IISTORE(IINTR)=I
381      CALL MARKEM(I)
382 1000 CONTINUE
383
384      RETURN
385      END
386
387
388      FUNCTION NABLED(NUMBER)
389
390 C      RETURN 1 IF TRANSITION # 'NUMBER' IS ENABLED, READY
391 C      TO FIRE. ELSE RETURN 0.
392
393      COMMON/TRANS/ITRANS(400,7),NXTTRN
394      COMMON/EVENT/IEVENT(400,6),NXT EVT
395      COMMON/IOTAB/IOTABL(4000),NXTIRE
396
397 C      CHECK LIST OF INPUTS TO SEE IF ALL ARE MARKED
398
399      MARK=0
400      IPT=ITRANS(NUMBER,2)
401      KOUNT=IOTABL(IPT)
402
403      DO 1000 I=IPT+1,IPT+KOUNT
404      NEVENT=IOTABL(I)
405      IF(IEVENT(NEVENT,2).GT.0) MARK=MARK+1
406 1000 CONTINUE
407
408      NABLED=0
409      IF(KOUNT.EQ.MARK) NABLED=1
410      RETURN
411      END
412
413
414      SUBROUTINE UNMARK(NUMBER,IERROR)
415 C
416 C      UNMARK (IE. DECREMENT THE NUMBER OF TOKENS
417 C      ALL OF THE INPUT EVENTS TO TRANSITION # 'NUMBER'.
418 C      RETURN IERROR=0 .....UNLESS.....
419 C      ONE EVENT IS A MULTIPLE INPUT OF THE SAME
420 C      TRANSITION AND WE DONT HAVE ENOUGH MARKERS.

```



```

421 C          WHEN THIS HAPPENS, REPLACE ANY REMOVED
422 C          TOKENS AND RETURN IERROR=1.
423 C
424          COMMON/EVENT/IEVENT(400,6),NXTEVT
425          COMMON/TRANS/ITRANS(400,7),NXTTRN
426          COMMON/IOTAB/IOTABL(4000),NXTTRE
427
428          IPT=ITRANS(NUMBER,2)
429          KOUNT=IOTABL(IPT)
430          IERROR=0
431
432          DO 1000 I=IPT+1,IPT+KOUNT
433          NEVENT=IOTABL(I)
434          J=I
435          IEVENT(NEVENT,2)=IEVENT(NEVENT,2)-1
436          IF(IEVENT(NEVENT,2).LT.0) GO TO 2000
437 1000      CONTINUE
438          RETURN
439
440 2000      CONTINUE
441          DO 3000 I=IPT+1,J
442          NEVENT=IOTABL(I)
443 3000      IEVENT(NEVENT,2)=IEVENT(NEVENT,2)+1
444          IERROR=1
445
446          RETURN
447          END
448
449
450          SUBROUTINE MARKEM(NUMBER)
451 C
452 C          MARK ALL OUTPUT EVENTS OF TRANSITION # 'NUMBER'
453 C
454          COMMON/EVENT/IEVENT(400,6),NXTEVT
455          COMMON/TRANS/ITRANS(400,7),NXTTRN
456          COMMON/IOTAB/IOTABL(4000),NXTTRE
457
458          IPT=ITRANS(NUMBER,3)
459          KOUNT=IOTABL(IPT)
460          DO 1000 I=IPT+1,IPT+KOUNT
461          NEVENT=IOTABL(I)
462 1000      IEVENT(NEVENT,2)=IEVENT(NEVENT,2)+1
463          RETURN
464          END
465
466          SUBROUTINE SCANR
467
468          BYTE IWORD,ISC,IBLANK
469          COMMON/SCAN/IWORD(15,10),NUMBER
470          BYTE NBUFFR
471          COMMON/SCAN1/NBUFFR(80)
472          DATA ISC/1H:/
473          DATA IBLANK/1H /
474 0001      FORMAT(80A1)
475          READ(1,0001,END=9999,ERR=9999) (NBUFFR(I),I=1,80)
476          IPOINT=1
477 C          SET POINTER TO FIRST CHARACTER IN THE BUFFER
478 C          NOW PROCESS THE FIRST 15 TOKENS DELIMITED BY EITHER
479 C          A BLANK (OR MULTIPLE BLANKS) OR A SEMICOLON.
480 C

```



```

481      DO 0002 NUMBER=1,15
482      IFLAG=0
483 C     SET IWORD(NUMBER,X)=IBLANK (SET WORD TO ALL BLANKS)
484      DO 0003 I=1,10
485 0003 IWORD(NUMBER,I)=IBLANK
486 C     START SCANNING LINE FROM POINTER ON TO FIND NON-BLANK
487      KOUNT=1
488 C     "KOUNT" KEEPS TRACK OF THE NO. OF CHAR. IN THE TOKEN
489      DO 0004 KPOINT=IPOINT,80
490      IF(NBUFFER(KPOINT).NE.IBLANK .AND. NBUFFER(KPOINT).NE.ISC)
491 1      GO TO 0005
492      IF(IFLAG.EQ.0) GO TO 0004
493      IF(IFLAG.EQ.1) GO TO 0006
494 0005 CONTINUE
495      IFLAG=1
496      IWORD(NUMBER,KOUNT)=NBUFFER(KPOINT)
497      KOUNT=KOUNT+1
498      IF(KOUNT.GT.10) GO TO 0006
499 0004 CONTINUE
500
501 0006 CONTINUE
502 C     END OF TOKEN FOUND, RESET SOME POINTERS
503      IPOINT=KPOINT+1
504      IF(IPOINT.GT.80) GO TO 0010
505
506 0002 CONTINUE
507 C     END OF BASIC TOKEN GETTING LOOP
508
509 0010 NUMBER=NUMBER-1
510      RETURN
511 9999 CONTINUE
512 C     END OF FILE OR I/O ERROR DETECTED
513 9998 FORMAT(' EOF OR ERROR ON SCANNER INPUT FROM UNIT 1')
514      TYPE 9998
515      NUMBER=0
516      RETURN
517      END
518
519
520
521      SUBROUTINE XINTGR(NWORD,IVALUE)
522
523 C          CONVERT THE ENTRY IN "IWORD" TO INTEGER
524 C          RETURN INTEGER "IVALUE"
525
526      BYTE IWORD
527      COMMON/SCAN/IWORD(15,10),NUMBER
528      BYTE TSTRNG
529      DIMENSION TSTRNG(10)
530      BYTE IBLANK
531      DATA IBLANK/1H /
532      DO 0001 I=1,10
533      KOUNT=I
534      TSTRNG(I)=IWORD(NWORD,I)
535      IF(IWORD(NWORD,I).EQ.IBLANK) GO TO 1000
536 0001 CONTINUE
537 1000 CONTINUE
538      KOUNT=KOUNT-1
539
540 2004 FORMAT(X,10A1)

```



```

541 2005 FORMAT(X,1I10)
542 DO 2006 I=1,KOUNT
543     J=11-I
544     K=(KOUNT+1)-I
545 2006 TSTRNG(J)=TSTRNG(K)
546     L=10-KOUNT
547 DO 2007 I=1,L
548 2007 TSTRNG(I)=IBLANK
549     OPEN(UNIT=2,NAME='ISTORE',TYPE='NEW',INITIALSIZE=20)
550     WRITE(2,2004) (TSTRNG(I),I=1,10)
551     CLOSE(UNIT=2,DISPOSE='SAVE')
552     OPEN(UNIT=2,NAME='ISTORE',TYPE='OLD')
553     READ(2,2005) IVALUE
554     CLOSE(UNIT=2,DISPOSE='DELETE')
555     RETURN
556 END
557
558
559 FUNCTION MATCHS(NUMB,STRING,NCHAR)
560
561 C      THIS FUNCTION DETERMINES IF SCANNER TOKEN
562 C      IWORD(NUMB) MATCHES THE CHARACTERS IN "STRING"
563 C      AT LEAST FOR THE FIRST "NCHAR" CHARACTERS.
564
565 C      IF THERE IS A MATCH, IT RETURNS THE INTEGER "1"
566 C      NO MATCH RETURNS "0".
567
568     BYTE IWORD
569     COMMON/SCAN/IWORD(15,10),NUMBER
570     BYTE STRING
571     DIMENSION STRING(10)
572     MATCHS=0
573
574     DO 0001 I=1,NCHAR
575     IF(IWORD(NUMB,I).NE.STRING(I)) RETURN
576 0001 CONTINUE
577
578 C      IF YOU GET HERE, THEY WERE THE SAME...
579     MATCHS=1
580     RETURN
581 END
582
583
584 C      END OF PROGRAM SIMULATOR

```



```

1 #
2
3 /*****
4 *****/
5 /*****
6 *****/
7 /*****
8 *****/
9 /*****
10 *****/
11 /*****
12 *****/
13 /*****
14 *****/
15 /*****
16 *****/
17
18
19 /*****/
20 /** EXTERNAL DECLARATIONS **/
21 /*****/
22
23
24 /*** LITERALS ***/
25
26 #define header 4
27 #define pictures 50
28 #define bounds 100
29 #define limit 500
30
31
32 /*** STRUCTURES ***/
33
34 struct { /* data structure information on net nodes .. */
35
36     int ctrl1; /* store control char not used in program ... */
37     int nameptr; /* index to names array ..... */
38     int marker; /* initial marker state of the network ..... */
39     int xcord; /* x coordinate of place ..... */
40     int ycord; /* y coordinate of place ..... */
41     int plot; /* whether or not place is to be plotted .... */
42     int length; /* length of name associated with place ..... */
43
44 }file1 [bounds], *bp1; /* pointer into data structure ..... */
45
46
47 struct { /* data structure information on transitions. */
48
49     int ctrl2; /* store control char not used in program ... */
50     int trnptr; /* index to names array ..... */
51     int inptr; /* pointer to inputs for a transition ..... */
52     int outptr; /* pointer to outputs for a transition ..... */
53     int xxcord; /* x coordinate of transition ..... */
54     int yycord; /* y coordinate of transition ..... */
55     int trnplot; /* whether or not transition is to be plotted */
56     int trnlen; /* length of name associated with transition */
57
58 }file2 [bounds], *bp2; /* pointer into data structure ..... */
59
60

```



```

61 /*** INTEGERS ***/
62
63 int a1,a2,a3,a4;          /* global storage for each data structure ... */
64 int buffer[bounds];      /* buffer into which each frame is read ..... */
65 int cntrl[1];            /* variable containing # transitions fired .. */
66 int ctroverflow;         /* keep track of nodes overflow status ..... */
67 int dflcolor;            /* a default color for indicating overflows . */
68 int fdfbuf;              /* file descriptor for RUN..Y files ..... */
69 int fdabuf;              /* file descriptor for RUN..Z files ..... */
70 int firing[bounds];      /* storage into which fired places are read . */
71 int ictr;                /* counter passed to a function ..... */
72 int ievents;             /* number of non- & displayable nodes ..... */
73 int iflag;               /* counter for the interrupt mechanism ..... */
74 int iotbl[limit];        /* forms input-to-output relationship ..... */
75 int kpictures;           /* counter for the iterations of the network. */
76 int linktbl[100][4];     /* version 1 & 2 screen nodes locations .... */
77 int nbrplot;             /* counter for number of displayed nodes .... */
78 int nbytes[2];           /* store count fields for data structures ... */
79 int overflowtbl[100][2]; /* data structure to store overflow locations */
80 int set;                 /* user selected conrac graphics screen ..... */
81 int tflag[20];           /* saves flag for later use by trnlite() .... */
82 int tblctr;              /* a counter for version 3..reset conditions. */
83 int uniquetbl[100][4];   /* reset table locations for version 3 ..... */
84 int vers;                /* user selected option ..... */
85
86
87 /*** CHARACTERS ***/
88
89 char fbuf[20];           /* buffer to store name of second file ..... */
90 char gbuf[20];           /* buffer to store name of third file..... */
91 char names[limit];       /* character array for node labels ..... */
92 char scrn;               /* option variable for display to the screen. */
93
94
95 /***** */
96 /*** FUNCTION MAIN ***/
97 /***** */
98
99 main() {
100
101     extern rubout();      /* declare 'rubout' globally ..... */
102     init();               /* read input file ..... */
103     determine();          /* verify if user wants to see data structure */
104     display();             /* display input to crt ..... */
105     select();              /* select version of simulation & denisco set */
106     prepare(2);           /* prepare denisco-conrac ..... */
107     drawnode();           /* draw network nodes on conrac ..... */
108     places();              /* verify correct nodes drawn ..... */
109     trnslink();           /* function displays network transitions .... */
110     imark();              /* starting status of network packets ..... */
111     signal(2,rubout);     /* sets 'BRK' as interrupt ..... */
112     markino();            /* successive iterations of network flow .... */
113     gnfini();             /* closing out graphics facilities ..... */
114
115 }
116
117
118 /***** */
119 /*** PROGRAM FUNCTIONS ***/
120 /***** */

```



```

121
122
123 pause(peroid) {
124     /* function necessary as sleep() not compatible with signal() */
125     int i,j,k;
126
127     printf("***>>interrupt.....");
128     for(i=0;i<peroid;i++) {
129         for(j=0;j<400;j++) {
130             for(k=0;k<1000;k++) {
131                 }
132             }
133         }
134     printf("***>>wait.....");
135     return;
136 }
137
138
139 rubout() {
140     /* function enables the 'brk' key as the interrupt signal */
141     char halt;
142
143     space(2);
144     printf
145     ("***>>> received signal...frame number %d...<ret> to continue \n",
146      (iflag+2));
147     printf
148     ("***>>> for termination of program...type 'brk' from console \n");
149     while ((halt=getchar())!='\n') {
150         /* do-nothing loop */
151     }
152     signal(2,rubout);
153     return;
154 }
155
156
157 space(returns) {
158     int i;
159
160     for (i=0; i<returns; i++) {
161         printf("\n");
162     }
163     return;
164 }
165
166
167 init() {
168     /* function opens unformatted file & initializes start condition */
169     int a,bufctr,count,fd,i,j;
170     char cbuf[20],c,f;
171
172     space(2);
173     printf("***->>TRANSGRAPH ILLUSTRATES PETRI NET SIMULATION MODELS");
174     space(2);
175     printf("***->>ENTER THE NAME OF THE FILE TO BE PROCESSED..BUT \n");
176     printf("      NOTE THAT THIS FILE MUST BE AN UNFORMATTED FILE \n");
177     printf("      PRODUCED AS A RESULT OF EXECUTING simulator.out \n");
178     printf("      THE LAST LETTER OF WHICH MUST END IN LETTER 'X' \n");
179     error:space(2);
180     printf("***->>");

```



```

181
182     i=0;
183     while((c=getchar()) != '\n') {
184         cbuf[i]=c;
185         i++;
186     }
187     cbuf[i]='\0';
188
189     bufctr=i;
190     for(j=0;j<bufctr;j++) {
191         qbuf[j]=fbuf[j]=cbuf[j];
192         if(cbuf[j]=='X') {
193             fbuf[j]='Y';
194             gbuf[j]='Z';
195             qbuf[j+1]=fbuf[j+1]='\0';
196             j=bufctr;
197         }
198     }
199
200     fd = open(cbuf,0);
201     if (fd <= 0) {
202         printf("***->error occurred in opening file...try again");
203         space(3);
204         goto error;
205     }
206
207     if((count = read(fd,nbytes,header)) != header)
208         printf("error occurred in nbytes read");
209     ievents = (nbytes[1] - 1);
210     a1 = nbytes[1];
211     a = (nbytes[1]-1)*14;
212     if((count = read(fd,file1,a)) != a)
213         printf("error occurred in file1 read");
214
215     if((count=read(fd,nbytes,header))!=header)
216         printf("error occurred in nbytes read");
217     a2=nbytes[1];
218     a=(nbytes[1]-1)*16;
219     if((count=read(fd,file2,a))!=a)
220         printf("error occurred in file2 read");
221
222     if((count=read(fd,nbytes,header))!=header)
223         printf("error occurred in header read");
224     a3=nbytes[1];
225     a=(nbytes[1]+1)*2;
226     if((count=read(fd,iotbl,a))!=a)
227         printf("error occurred in iotbl read");
228
229     if((count=read(fd,nbytes,header))!=header)
230         printf("error occurred in header read");
231     a4=nbytes[1];
232     a=nbytes[1]+1;
233     if((count=read(fd,names,a))!=a)
234         printf("error occurred in names read");
235
236     close(fd);
237     return;
238 }
239
240

```



```

241 determine() {
242     int i;
243     char d,dbuf[20];
244
245     space(2);
246     printf("***->FUNCTION 'DETERMINE' ALLOWS THE USER \n");
247     printf("        TO EXAMINE ALL PRIMARY DATA STRUCTURES");
248 over:space(2);
249     printf("***->IF THIS FEATURE IS DESIRED TYPE 1 IF NOT 0,...<RET>");
250     space(2);
251     printf("***->");
252
253     i=0;
254     while((d=getchar())!='\n') {
255         dbuf[i]=d;
256         i++;
257     }
258     dbuf[i]='\0';
259
260     i=0;
261     while(dbuf[i]!='\0') {
262         d=dbuf[i];
263         switch(d) {
264             case '0':
265                 scrn='0';
266                 break;
267             case '1':
268                 scrn='1';
269                 printf("***->USE CONTROL Q WHEN SCREEN FULL");
270                 break;
271             default:
272                 printf("***->either blank or invalid entry");
273                 goto over;
274                 break;
275         }
276         i++;
277     }
278     return;
279 }
280
281
282 display() {
283     int i;
284
285     if(scrn=='1') {
286         space(2);
287         bp1 = file1;
288         printf("***-> FILE1 DATA STRUCTURE");
289         space(2);
290         printf("\nfeed    #    marker    xcord    ycord    plot    length \n");
291         for (i=0;i<a1;i++){
292             printf("%d \t %d \t %d \t %d \t %d \t %d \t %d \t \n",
293                 bp1->ctrl1, bp1->nameptr, bp1->marker, bp1->xcord,
294                 bp1->ycord, bp1->plot, bp1->length);
295             bp1++;
296         }
297
298         space(2);
299         bp2 = file2;
300         printf("***-> FILE2 DATA STRUCTURE");

```



```

301     space(2);
302     printf
303     ("Infeed   trnptr   intrn   outtrn   xxcord   yycord   trnplot   trnlen \n");
304     for (i=0;i<a2; i++){
305         printf("%d \t %d \t %d \t %d \t %d \t %d \t %d \t %d \t \n",
306             bp2->ctrl2, bp2->trnptr, bp2->intrn, bp2->outtrn,
307             bp2->xxcord, bp2->yycord, bp2->trnplot, bp2->trnlen);
308         bp2++;
309     }
310
311     space(2);
312     printf("***-> IOTBL DATA ARRAY");
313     space(2);
314     for (i=0; i<a3; i++) {
315         printf("%d ", iotbl[i]);
316     }
317
318     space(2);
319     printf("***-> NAMES DATA ARRAY");
320     space(2);
321     for (i=0; i<a4; i++) {
322         printf("%c", names[i]);
323     }
324     space(2);
325 }
326 return;
327 }
328
329
330 prepare(type) {
331     /* function designates set, screen size and color table */
332     int n,t,y;
333
334     y=0;
335     denisco (set);
336     erase();
337     screen(0.0,0.0,511.0,511.0);
338     setmod(type);
339     coltab();
340     colort(11);
341     for(n=12;n<14;n++) {
342         color(n);
343         for(t=0;t<512;t++) {
344             segmnt(0,y+t,511,y+t);
345         }
346     }
347     return;
348 }
349
350
351 drawnode() {
352     /* function displays type & location of network nodes */
353     char c,*nptr,hold;
354     int a,b,clrlbl,count,d,entry,h,i,j,k,l,test,x,y,*z;
355     float s;
356     double sqrt();
357
358     bol = file1;
359     a=0;
360     count = 1;

```



```

361
362     while((test=(bp1->nameptr))!=0) {
363         color(14);
364         test++;
365         z = &names[test];
366         c = *z;
367
368         if ((h=(bp1->plot))!= 0) {
369             switch (c){
370                 case 'I':
371                     x = (bp1->xcord);
372                     y = (bp1->ycord);
373                     for(d=0;d<10;d++) {
374                         segmnt(x-16,y+2+d,x,y+2+d);
375                     }
376                     linktbl[al][3]=1;
377                     break;
378                 case 'O':
379                     x = (bp1->xcord);
380                     y = (bp1->ycord);
381                     for(d=0;d<10;d++) {
382                         segmnt(x-16,y-2-d,x,y-2-d);
383                     }
384                     linktbl[a][3]=1;
385                     break;
386                 case 'R':
387                     x = (bp1->xcord);
388                     y = (bp1->ycord);
389                     for(d=0;d<31;d++) {
390                         segmnt
391                         (x-18+d/4,y+15-d,x+18-d/4,y+15-d);
392                     }
393                     clr1bl=14;
394                     label(x,y,test,clr1bl);
395                     break;
396                 case 'S':
397                     x = (bp1->xcord);
398                     y = (bp1->ycord);
399                     for(d=0;d<31;d++) {
400                         segmnt(x-18,y-15+d,x+18,y-15+d);
401                     }
402                     if(vers==3) {
403                         nptr= &names[test+1];
404                         hold= *nptr;
405                         clr1bl=atoi(&hold);
406                         label(x,y,test,clr1bl);
407                         color(14);
408                     }
409                     else {
410                         clr1bl=14;
411                         label(x,y,test,clr1bl);
412                     }
413                     break;
414                 case 'I':
415                     x = (bp1->xcord);
416                     y = (bp1->ycord);
417                     for (k=0;k<19;k++) {
418                         s=k;
419                         i=(x-(sqrt(324.-s*s)));
420                         j=y+s;

```



```

421             l=(x+(sqrt(324.-s*s)));
422             seamnt(i,j,l,j);
423         }
424         for (k=0;k<19;k++) {
425             s=k;
426             i=(x-(sqrt(324.-s*s)));
427             j=y-s;
428             l=(x+(sqrt(324.-s*s)));
429             seamnt(i,j,l,j);
430         }
431         if(vers==3) {
432             nptr = &names[test+1];
433             hold = *nptr;
434             clrbl=atoi(&hold);
435             label(x,y,test,clrbl);
436             color(14);
437         }
438         else {
439             clrbl=14;
440             label(x,y,test,clrbl);
441         }
442         break;
443     default:
444         printf("name not valid identifier");
445         space(2);
446         break;
447     }
448
449     linktbl[a][0]=count;
450     linktbl[a][1] = x;
451     linktbl[a][2] = y;
452     a++;
453 }
454 count++;
455 bpl++;
456 }
457 if(vers==3) {
458     entry=0;
459     uniquetbl[entry][0]=linktbl[0][0];
460     uniquetbl[entry][1]=linktbl[0][1];
461     uniquetbl[entry][2]=linktbl[0][2];
462     uniquetbl[entry][3]=linktbl[0][3];
463     tblctr=1;
464     for(i=0;i<a;i++) {
465         if(uniquetbl[entry][1]==linktbl[i+1][1] &&
466            uniquetbl[entry][2]==linktbl[i+1][2]) {
467             /* do nothing */
468         }
469         else {
470             entry++;
471             uniquetbl[entry][0]=linktbl[i+1][0];
472             uniquetbl[entry][1]=linktbl[i+1][1];
473             uniquetbl[entry][2]=linktbl[i+1][2];
474             uniquetbl[entry][3]=linktbl[i+1][3];
475             tblctr++;
476         }
477     }
478 }
479 nbrofot=a;
480 return;

```



```

481 }
482
483
484 places() {
485     int h,i,j,k;
486
487     if(strn=='1') {
488         space(2);
489         printf("***->DATA STRUCTURE LINKTBL ");
490         space(2);
491         for(i=0;i<nbrplot;i++) {
492             for(h=0;h<4;h++) {
493                 printf("%d --",linktbl[i][h]);
494             }
495             space(1);
496         }
497         space(2);
498         if(vers==3) {
499             printf("***->DATA STRUCTURE UNIQUELBL");
500             space(2);
501             for(j=0;j<tblctr;j++) {
502                 for(k=0;k<4;k++) {
503                     printf("%d --",uniquetbl[j][k]);
504                 }
505                 space(1);
506             }
507         }
508         space(2);
509     }
510     return;
511 }
512
513
514 label(xx,yy,zz,clbl) {
515     /* determines node label placement in relation to node */
516     int f,i;
517     char a;
518
519     color(clbl);
520     if(xx>250) {
521         if(yy>250) a='1';
522         else a='2';
523     }
524     else {
525         if(yy>250) a='3';
526         else a='4';
527     }
528     f=(bol->length);
529
530     switch (a) {
531         case '1':
532             for(i=0;i<2;i++) {
533                 charac((xx+(20+(8*i))),yy+15,names[zz]);
534                 zz++;
535             }
536             break;
537         case '2':
538             for(i=0;i<2;i++) {
539                 charac((xx+(20+8*i)),yy-28,names[zz]);
540                 zz++;

```



```

541         }
542         break;
543     case '3':
544         for(i=0;i<2;i++) {
545             charac((xx-(32-(8*i))),yy+15,names[zz]);
546             zz++;
547         }
548         break;
549     case '4':
550         for (i=0;i<2;i++) {
551             charac((xx-(32-8*i)),yy-26,names[zz]);
552             zz++;
553         }
554         break;
555     }
556 return;
557 }
558
559
560 select() {
561     int i,n;
562     char v,vbuf[20];
563
564     space(2);
565     printf("***->THERE ARE 3 VERSIONS TO THIS GRAPHICS PACKAGE \n");
566     printf("PLEASE SELECT ONE OF THE FOLLOWING VERSIONS: \n");
567     again:space(2);
568     printf("VERSION 1 ... PETRI-NET PACKAGE ..... TYPE 1 \n");
569     printf("VERSION 2 ... PACKET REPRESENTATION ... TYPE 2 \n");
570     printf("VERSION 3 ... MULTIROUTING PACKAGE .... TYPE 3 \n");
571     n = 0;
572     twice:space(2);
573     printf("***->");
574     n++;
575     i = 0;
576     while ((v=getchar()) != '\n') {
577         vbuf[i] = v;
578         i++;
579     }
580     vbuf[i] = '\0';
581
582     i = 0;
583     while(vbuf[i] != '\0') {
584         v = vbuf[i];
585         if(n==1) {
586             switch(v) {
587                 case '1':
588                     vers = 1;
589                     break;
590                 case '2':
591                     vers = 2;
592                     break;
593                 case '3':
594                     vers = 3;
595                     break;
596                 default:
597                     printf
598                     ("***->incorrect version try again!");
599                     goto again;
600             }

```



```

601         }
602         i++;
603     }
604     else {
605         switch(v) {
606             case '0':
607                 set = 0;
608                 break;
609             case '1':
610                 set = 1;
611                 break;
612             case '2':
613                 set = 2;
614                 break;
615             default:
616                 printf
617                 ("***->incorrect genisco set try again!");
618                 printf
619                 ("      set selection should be 0,1 or 2");
620                 n = 1;
621                 goto twice;
622                 break;
623         }
624         i++;
625     }
626 }
627 if(n==1) {
628     space(2);
629     printf("***->NOW SELECT THE GENISCO SET YOU WISH \n");
630     printf("      THE PROGRAM TO BE DISPLAYED TO..... \n");
631     printf("      IN C3 LAB EITHER SET0, SET1 OR SET2 \n");
632     goto twice;
633 }
634 return;
635 }
636
637
638 imark() {
639     /* marks initial state of system by calling appropriate function */
640     int b,colour,era,x,y;
641
642     bnl = file1;
643     dfltcolor=3;
644     color(dfltcolor);
645     colour=2;
646     ctroverflow=0;
647     while ((c=(bnl->nameptr)) !=0) {
648         if((b=(bnl->plot)) !=0) {
649             switch(vers) {
650                 case 1:
651                     ivers1();
652                     break;
653                 case 2:
654                     ivers2(colour);
655                     break;
656                 case 3:
657                     ivers3(colour);
658                     break;
659             }
660         }

```



```

661         if(dfltcolor!=3) color(3);
662         bol++;
663     }
664     color(14);
665     printf(0,350.,480., "TIME FRAME = 1");
666     preread(1);
667     trnlite(1);
668     disola();
669     hold();
670     trnlite(2);
671     reset(1);
672     color(13);
673     overflow();
674     printf(0,350.,480., "TIME FRAME = 1");
675     return;
676 }
677
678
679 hold() {
680     int holding;
681
682     space(2);
683     printf("***->THIS IS THE INITIAL STATE OF THE NETWORK \n");
684     printf("      TYPE <RETURN> TO CONTINUE EXECUTION..... \n");
685     while((holding=getchar())!='\n') {
686         /* do nothing loop */
687     }
688     return;
689 }
690
691
692 ivers1() {
693     int e,x,y,z;
694     char check;
695
696     e=(bol->marker);
697     x=(bol->xcord);
698     v=(bol->ycord);
699     z=(bol->nameptr);
700     if((check=names[z+1])!='I' && (check=names[z+1])!='O') {
701         printf(0,x-3.0,511.-(y-3), "%d",e);
702     }
703     else {
704         if(check=names[z+1]=='I') printf(0,x-14.0,511.-(y+2), "%d",e);
705         else printf(0,x-14.0,511.-(y-9), "%d",e);
706     }
707     return;
708 }
709
710
711 ivers2(colour) {
712     int e,x,v,z;
713     char check;
714
715     e=(bol->marker);
716     x=(bol->xcord);
717     v=(bol->ycord);
718     z=(bol->nameptr);
719     if((check=names[z+1])!='I' && (check=names[z+1])!='O') {
720         pckt2(x,v,e,colour);

```



```

721     }
722     else {
723         if(check=names[z+1]=='I') printf(0,x-14.0,511.-(y+2),"%a",e);
724         else printf(0,x-14.0,511.-(y-9),"%d",e);
725     }
726     return;
727 }
728
729
730 ivers3(colour) {
731     char keep,*kptr;
732     int a,aa,b,bb,c,cc,clr[25],i,k,n,stack,total,x,y;
733
734     total=0;
735     n=1;
736     x=(bpl->xcord);
737     y=(bpl->ycord);
738     a=(bpl->length);
739     b=(bpl->nameptr);
740     c=(bpl->marker);
741     total=total+c;
742
743     if(c>0) {
744         for(i=0;i<c;i++) {
745             kptr = &names[b+a-2];
746             keep = *kptr;
747             clr[n]=atoi(&keep);
748             n++;
749         }
750     }
751
752     kptr = &names[b+a];
753     keep = *kptr;
754     stack=atoi(&keep);
755
756     if(names[b+(a-1)]!='0') {
757         if(names[c+(a-1)]=='2') stack=stack+20;
758         else stack = stack + 10;
759     }
760
761     for(i=0;i<stack-1;i++) {
762         bpl++;
763         aa=(bpl->length);
764         bb=(bpl->nameptr);
765         cc=(bpl->marker);
766         total=total+cc;
767         if(cc>0) {
768             for(k=0;k<cc;k++) {
769                 kptr = &names[bb+aa-2];
770                 keep = *kptr;
771                 clr[n]=atoi(&keep);
772                 n++;
773             }
774         }
775     }
776     if(names[b+1]!='I' && names[b+1]!='0') {
777         nckt3
778         (x,y,total,clr[1],clr[2],clr[3],clr[4],clr[5],clr[6],clr[7],
779         colour);
780     }

```



```

781     else {
782         if(names[b+1]!='I') printf(0,x-14.0,511.-(y+2),"%d",total);
783         else printf(0,x-14.0,511.-(y-9),"%d",total);
784     }
785 return;
786 }
787
788
789 preread(flag) {
790     int bucket[2],count,fd,fa,i,nbrtrns;
791     if(flag!=3) {
792         if(flag==1) {
793             fd = open (fbuf, 0);
794             if (fd<=0) {
795                 printf("***->error occurred in opening fd file");
796             }
797             fdfbuf=fd;
798             fa = open (abuf, 0);
799             if (fa<=0) {
800                 printf("***->error occurred in opening fg file");
801             }
802             fdqbuf=fg;
803         }
804         if((count=read (fdfbuf, bucket, 2))!=2) {
805             printf("***->error occurred in fd bucket read");
806         }
807         if((count=read (fdfbuf, buffer,(ievents*2)))!=(ievents*2)) {
808             printf("***->error occurred in buffer read");
809         }
810         if((count=read (fdqbuf, bucket, 2))!=2) {
811             printf("***->error occurred in fa bucket read");
812         }
813         if((count=read (fdqbuf, cntnl,2))!=2) {
814             printf("***->error occurred in cntnl read");
815         }
816         if(cntnl[0]==0) {
817             soace(2);
818             printf("***->the last network state has been achieved");
819             koictures=pictures+1;
820             soace(2);
821         }
822     } else {
823         if((count=read (fdqbuf, bucket, 2))!=2) {
824             printf("***->error occurred in bucket read");
825         }
826         ntrtrns = cntnl[0]*2;
827         if((count=read (fdqbuf, firing,nbrtrns))!= nbrtrns) {
828             printf("***->error occurred in firing read");
829         }
830     }
831 }
832 else {
833     close(fdfbuf);
834     close(fdqbuf);
835 }
836 return;
837 }
838
839
840 stage() {

```



```

841
842     if (kpictures!=0 && kpictures<pictures+1) {
843         preread(2);
844         if(kpictures!=pictures+1) {
845             trnlite(1);
846             displa();
847             pause(1);
848             trnlite(2);
849         }
850     }
851     return;
852 }
853
854
855 marking() {
856     /* function displays successive iterations of the network */
857     int colour,draw,i,mark,n,x,y;
858
859     bpl = file1;
860     n = 2;
861
862     /* following loop processes ievent # oara entries each pass */
863     for(kpictures=0;kpictures<pictures;kpictures++) {
864         stage();
865         if(kpictures>0) {
866             reset();
867             color(13);
868             overflow();
869             printf(0,350.,480.,"TIME FRAME = %d",n);
870             n++;
871         }
872         iflag = kpictures;
873         draw = (bpl->plot);
874         dfltcolor=3;
875         color (dfltcolor);
876         colour=2;
877         ctroverflow=0;
878         for (i=0; i < ievents; i++) {
879             if (draw == 1) {
880                 ictr = i;
881                 switch (vers) {
882                     case 1:
883                         vers1();
884                         break;
885                     case 2:
886                         vers2(colour);
887                         break;
888                     case 3:
889                         vers3(colour);
890                         break;
891                 }
892             }
893             i = ictr++;
894             if(dfltcolor!=3) color(3);
895             bpl++;
896             draw = (bpl -> plot);
897         }
898         dfltcolor=14;
899         color(dfltcolor);
900         colour=13;

```



```

901         printf(0,350.,480., "TIME FRAME = %d",n);
902         pause(2);
903         trnlite(3);
904         bpl = file1;
905     }
906     preread(3);
907     return;
908 }
909
910
911 overflow() {
912
913     int i,x,y;
914
915     i=0;
916     while(overflowtbl[i][0]!=0) {
917         x=overflowtbl[i][0];
918         y=overflowtbl[i][1];
919         block((x-3)*1.,511.-(y+28)*1.,(x+10)*1.,511.-(y+20)*1.);
920         overflowtbl[i][0]=0;
921         overflowtbl[i][1]=0;
922         i++;
923     }
924     return;
925 }
926
927
928 vers1() {
929     int e,x,y,z;
930     char check;
931
932     color(3);
933     e=buffer[ictr];
934     x=(bol->xcord);
935     y=(bol->ycord);
936     z=(bol->nameptr);
937     if((check=names[z+1])!='I' && (check=names[z+1])!='O') {
938         printf(0,x-3.0,511.-(y-3),"%d",e);
939     }
940     else {
941         if(check=names[z+1]=='I') printf(0,x-14.0,511.-(y+2),"%d",e);
942         else printf(0,x-14.0,511.-(y-9),"%d",e);
943     }
944     return;
945 }
946
947
948 vers2(colour) {
949     int mark,x,y,z;
950     char check;
951
952     x=(bol->xcord);
953     y=(bol->ycord);
954     z=(bol->nameptr);
955     mark=buffer[ictr];
956     if((check=names[z+1])!='I' && (check=names[z+1])!='O') {
957         ockt2(x,y,mark,colour);
958     }
959     else {
960         if((check=names[z+1]=='I') printf(0,x-14.0,511.-(y+2),"%d",mark);

```



```

961     else printf(0,x-14.0,511.-(y-9),"%d",mark);
962 }
963 return;
964 }
965
966
967 vers3(colour) {
968     int a,aa,b,bb,mark,marks,clr[25],j,k,n,stack,total,x,y;
969     char keep,*kptr;
970
971     total=0;
972     n=1;
973     x=(bp1->xcord);
974     y=(bp1->ycord);
975     a=(bp1->length);
976     b=(bp1->nameptr);
977     mark=buffer[ictr];
978     total=total+mark;
979     if(mark>0) {
980         for(j=0;j<mark;j++) {
981             kptr = &names[b+a-2];
982             keep = *kptr;
983             clr[n]=atoi(&keep);
984             n++;
985         }
986     }
987
988     kptr = &names[b+a];
989     keep = *kptr;
990     stack=atoi(&keep);
991
992     if(names[b+(a-1)]!='0') {
993         if(names[b+(a-1)]=='2') stack=stack+20;
994         else stack = stack + 10;
995     }
996
997     for(j=0;j<stack-1;j++) {
998         bp1++;
999         ictr++;
1000         aa=(bp1->length);
1001         bb=(bp1->nameptr);
1002         marks=buffer[ictr];
1003         total=total+marks;
1004         if(marks>0) {
1005             for(k=0;k<marks;k++) {
1006                 kptr = &names[bb+aa-2];
1007                 keep = *kptr;
1008                 clr[n]=atoi(&keep);
1009                 n++;
1010             }
1011         }
1012     }
1013     if(names[b+1]!='I' && names[b+1]!='0') {
1014         pkt3
1015         (x,y,total,clr[1],clr[2],clr[3],clr[4],clr[5],clr[6],clr[7],
1016         colour);
1017     }
1018     else {
1019         if(names[b+1]=='I') printf(0,x-14.0,511.-(y+2),"%d",total);
1020         else printf(0,x-14.0,511.-(y-9),"%d",total);

```



```

1021     }
1022     return;
1023 }
1024
1025
1026 ockt2(xaxis,yaxis,point,class) {
1027
1028     switch(point) {
1029         case 0:
1030             break;
1031         case 1:
1032             block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1033             break;
1034         case 2:
1035             block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1036             block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1037             break;
1038         case 3:
1039             block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1040             block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1041             block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1042             break;
1043         case 4:
1044             block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1045             block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1046             block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1047             block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1048             break;
1049         case 5:
1050             block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1051             block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1052             block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1053             block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1054             block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
1055             break;
1056         case 6:
1057             block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1058             block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1059             block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1060             block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1061             block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
1062             block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
1063             break;
1064         case 7:
1065             block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1066             block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1067             block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1068             block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1069             block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
1070             block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
1071             block((xaxis+7)*1.,511.-(yaxis+13)*1.,(xaxis+13)*1.,511.-(yaxis+7)*1.);
1072             break;
1073         default:
1074             overflowtbl[ctroverflow][0]=xaxis;
1075             overflowtbl[ctroverflow][1]=yaxis;
1076             ctroverflow++;
1077             block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1078             block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1079             block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1080             block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);

```



```

1081     block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
1082     block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
1083     block((xaxis+7)*1.,511.-(yaxis+13)*1.,(xaxis+13)*1.,511.-(yaxis+7)*1.);
1084     color(class);
1085     dfltcolor=class;
1086     printf(0,xaxis-3.0,511.-(yaxis+22)+3.0,"%d",point-7);
1087     break;
1088 }
1089 return;
1090 }
1091
1092
1093
1094 ockt3(xaxis,yaxis,total,c1,c2,c3,c4,c5,c6,c7,class) {
1095
1096     switch (total) {
1097     case 0:
1098         break;
1099     case 1:
1100         color(c1);
1101         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1102         break;
1103     case 2:
1104         color(c1);
1105         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1106         color(c2);
1107         block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1108         break;
1109     case 3:
1110         color(c1);
1111         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1112         color(c2);
1113         block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1114         color(c3);
1115         block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1116         break;
1117     case 4:
1118         color(c1);
1119         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1120         color(c2);
1121         block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1122         color(c3);
1123         block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1124         color(c4);
1125         block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1126         break;
1127     case 5:
1128         color(c1);
1129         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1130         color(c2);
1131         block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1132         color(c3);
1133         block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1134         color(c4);
1135         block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1136         color(c5);
1137         block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
1138         break;
1139     case 6:
1140         color(c1);

```



```

1141     block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1142     color(c2);
1143     block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1144     color(c3);
1145     block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1146     color(c4);
1147     block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1148     color(c5);
1149     block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
1150     color(c6);
1151     block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
1152     break;
1153 case 7:
1154     color(c1);
1155     block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1156     color(c2);
1157     block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1158     color(c3);
1159     block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1160     color(c4);
1161     block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1162     color(c5);
1163     block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
1164     color(c6);
1165     block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
1166     color(c7);
1167     block((xaxis+7)*1.,511.-(yaxis+13)*1.,(xaxis+13)*1.,511.-(yaxis+7)*1.);
1168     break;
1169 default:
1170     overflowtbl[ctroverflow][0]=xaxis;
1171     overflowtbl[ctroverflow][1]=yaxis;
1172     ctroverflow++;
1173     color(c1);
1174     block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1175     color(c2);
1176     block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1177     color(c3);
1178     block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1179     color(c4);
1180     block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1181     color(c5);
1182     block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
1183     color(c6);
1184     block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
1185     color(c7);
1186     block((xaxis+7)*1.,511.-(yaxis+13)*1.,(xaxis+13)*1.,511.-(yaxis+7)*1.);
1187     color(class);
1188     dfltcolor=class;
1189     printf(0,xaxis-3.0,511.-(yaxis+22)+3.0,"%d",total-7);
1190     break;
1191 }
1192 return;
1193 }
1194
1195
1196 reset() {
1197     /* reset function for successive network iterations */
1198     int i,mark,x,y,z;
1199
1200     if(vers==1 || vers==2) {

```



```

1201     for(i=0;i<nbrplot;i++) {
1202         color(14);
1203         x=linktbl[i][1];
1204         y=linktbl[i][2];
1205         z=linktbl[i][3];
1206         if(z==0) {
1207             block((x-3)*1.,511.-(y-7)*1.,(x+3)*1.,511.-(y-13)*1.);
1208             block((x-13)*1.,511.-(y+13)*1.,(x+13)*1.,511.-(y-3)*1.);
1209         }
1210         else {
1211             block((x-16)*1.,511.-(y-2)*1.,x*1.,511.-(y-10)*1.);
1212             block((x-16)*1.,511.-(y+10)*1.,x*1.,511.-(y+2)*1.);
1213         }
1214     }
1215 }
1216 else {
1217     for(i=0;i<tblctr-1;i++) {
1218         color(14);
1219         x=uniquetbl[i][1];
1220         y=uniquetbl[i][2];
1221         z=uniquetbl[i][3];
1222         if(z==0) {
1223             block((x-3)*1.,511.-(y-7)*1.,(x+3)*1.,511.-(y-13)*1.);
1224             block((x-13)*1.,511.-(y+13)*1.,(x+13)*1.,511.-(y-3)*1.);
1225         }
1226         else {
1227             block((x-16)*1.,511.-(y-2)*1.,x*1.,511.-(y-10)*1.);
1228             block((x-16)*1.,511.-(y+10)*1.,x*1.,511.-(y+2)*1.);
1229         }
1230     }
1231 }
1232 return;
1233 }
1234
1235
1236 trn_lite(tflag) {
1237     int flag,h,i,inp,j,k,l,m,n,on,outp,ptr,r,s,x,xx,y,yy;
1238
1239     for(h=0;h<cntrl[0];h++) {
1240         bp2=file2;
1241         for(j=0;j<(firina[h]-1);j++) {
1242             bp2++;
1243         }
1244         if(on=(bp2->trnplot)==1) {
1245             x=(bp2->xxcord);
1246             y=(bp2->vvcord);
1247             if(tflag==1 || tflag==2) color(11);
1248             else color(15);
1249             segmnt(x,y-20,x,y+20);
1250             l=(bp2->trnlen);
1251             k=(bp2->trnptr);
1252             for(i=0;i<l;i++) {
1253                 charac((x-8+(8*i)),y+24,names[k+i]);
1254                 k++;
1255             }
1256
1257             if(tflag==1 || tflag==3) {
1258                 if(tflag==1) color(11);
1259                 else color(14);
1260                 flag=0;

```



```

1261     m=(bo2->intrn);
1262     r=iobl[m];
1263     for(i=0;i<r;i++) {
1264         ino=iobl[m+1];
1265         for(j=0;j<nbrplot;j++) {
1266             if(ino==linktbl[j][0]) {
1267                 xx=linktbl[j][1];
1268                 yy=linktbl[j][2];
1269                 if(flag==0) {
1270                     if(xx<=x) flag=1;
1271                     else flag=2;
1272                     sflag[h]=flag;
1273                 }
1274                 if(flag==1) intrns1(xx,yy,x,y);
1275                 else intrns2(xx,yy,x,y);
1276                 j=nbrplot;
1277             }
1278         }
1279         m++;
1280     }
1281 }
1282
1283     if(tflag==2 || tflag==3) {
1284         if(tflag==2) color(11);
1285         else color(14);
1286         n=(bo2->outtrn);
1287         s=iobl[n];
1288         for(i=0;i<s;i++) {
1289             outo=iobl[n+1];
1290             for(j=0;j<nbrplot;j++) {
1291                 if(outo==linktbl[j][0]) {
1292                     xx=linktbl[j][1];
1293                     yy=linktbl[j][2];
1294                     if(sflag[h]==1) outrns1(xx,yy,x,y);
1295                     else outrns2(xx,yy,x,y);
1296                     j=nbrplot;
1297                 }
1298             }
1299             n++;
1300         }
1301     }
1302 }
1303 }
1304 return;
1305 }
1306
1307
1308 trnslink() {
1309     int flag,i,ino,j,k,l,m,n,on,outo,ptr,r,s,x,xx,v,yy;
1310
1311     bo2=file2;
1312     while(ptr=(bo2->trnptr)!=0) {
1313         if(on=(bo2->trnplot)==1) {
1314             x=(bo2->xxcord);
1315             v=(bo2->yvcord);
1316             color(15);
1317             segmnt(x,y-20,x,y+20);
1318             l=(bo2->trnlen);
1319             k=(bo2->trnptr);
1320             for(i=0;i<l;i++) {

```



```

1321         charac((x-8+(8*i)),y+24,names[k+1]);
1322         k++;
1323     }
1324
1325     color(14);
1326     flag=0;
1327     m=(bo2->intern);
1328     r=iotbl[m];
1329     for(i=0;i<r;i++) {
1330         inp=iotbl[m+1];
1331         for(j=0;j<nbrolot;j++) {
1332             if(inp==linktbl[j][0]) {
1333                 xx=linktbl[j][1];
1334                 yy=linktbl[j][2];
1335                 if(flag==0) {
1336                     if(xx<=x) flag=1;
1337                     else flag=2;
1338                 }
1339                 if(flag==1) intrns1(xx,yy,x,y);
1340                 else intrns2(xx,yy,x,y);
1341                 j=nbrolot;
1342             }
1343         }
1344         m++;
1345     }
1346
1347     n=(bo2->outtern);
1348     s=iotbl[n];
1349     for(i=0;i<s;i++) {
1350         outp=iotbl[n+1];
1351         for(j=0;j<nbrolot;j++) {
1352             if(outp==linktbl[j][0]) {
1353                 xx=linktbl[j][1];
1354                 yy=linktbl[j][2];
1355                 if(flag==1) outrns1(xx,yy,x,y);
1356                 else outrns2(xx,yy,x,y);
1357                 j=nbrolot;
1358             }
1359         }
1360         n++;
1361     }
1362 }
1363 bo2++;
1364 }
1365 return;
1366 }
1367
1368 intrns1(xx,yy,x,y) {
1369     int input;
1370
1371     if(xx<(x-12)) {
1372         if(yy<=y) {
1373             if(yy==y) input=0;
1374             else input=1;
1375         }
1376         else input=2;
1377     }
1378 }
1379 else {
1380     if(yy<=y) input=4;

```



```

1381         else inout=3;
1382     }
1383
1384     switch (input) {
1385         case 0:
1386             segmnt(xx+16,yy,x-1,y);
1387             rhtarrow(x-1,y);
1388             break;
1389         case 1:
1390             segmnt(xx+12,yy+12,x-12,y-8);
1391             segmnt(x-12,y-8,x-1,y-8);
1392             rhtarrow(x-1,y-8);
1393             break;
1394         case 2:
1395             segmnt(xx+12,yy-12,x-12,y+8);
1396             segmnt(x-12,y+8,x-1,y+8);
1397             rhtarrow(x-1,y+8);
1398             break;
1399         case 3:
1400             segmnt(xx-12,yy-12,x-12,y+36);
1401             segmnt(x-12,y+16,x-1,y+16);
1402             lftsemi(x-12,y+26);
1403             rhtarrow(x-1,y+16);
1404             break;
1405         case 4:
1406             segmnt(xx-12,yy+12,x-12,y-36);
1407             segmnt(x-12,y-16,x-1,y-16);
1408             lftsemi(x-12,y-26);
1409             rhtarrow(x-1,y-16);
1410             break;
1411     }
1412     return;
1413 }
1414
1415
1416 rhtarrow(x,v) {
1417
1418     segmnt(x-4,y-4,x,y);
1419     segmnt(x-4,y+4,x,y);
1420     return;
1421 }
1422
1423
1424 lftsemi(x,v) {
1425     int i,j,k;
1426     double sqrt();
1427
1428     for(k=0;k<11;k++){
1429         i=(v-(sqrt(100.-k*k)));
1430         j=(v+(sqrt(100.-k*k)));
1431         dot(x-(k/2.),511.-i);
1432         dot(x-(k/2.),511.-j);
1433         if(k>8) {
1434             dot(x-(k/2.),511.-i-2);
1435             dot(x-(k/2.),511.-i-1);
1436             dot(x-(k/2.),511.-i+1);
1437             dot(x-(k/2.),511.-i+2);
1438             dot(x-(k/2.),511.-j-2);
1439             dot(x-(k/2.),511.-j-1);
1440             dot(x-(k/2.),511.-j+1);

```



```

1441         dot(x-(k/2.),511.-j+2);
1442     }
1443 }
1444 return;
1445 }
1446
1447
1448 outrnsi(xx,yy,x,y) {
1449     int output;
1450
1451     if(xx>(x+12)) {
1452         if(yy<=y) {
1453             if(yy==y) output=0;
1454             else output=1;
1455         }
1456         else output=2;
1457     }
1458     else {
1459         if(yy<=y) output=4;
1460         else output=3;
1461     }
1462
1463     switch (output) {
1464     case 0:
1465         segmnt(x+1,y,xx-20,yy);
1466         rhtarrow(xx-20,yy);
1467         break;
1468     case 1:
1469         segmnt(x+12,y-8,xx-28,yy+8);
1470         segmnt(xx-28,yy+8,xx-20,yy+8);
1471         rhtarrow(xx-20,yy+8);
1472         segmnt(x+1,y-8,x+12,y-8);
1473         break;
1474     case 2:
1475         segmnt(x+12,y+8,xx-28,yy-8);
1476         segmnt(xx-28,yy-8,xx-20,yy-8);
1477         rhtarrow(xx-20,yy-8);
1478         segmnt(x+1,y+8,x+12,y+8);
1479         break;
1480     case 3:
1481         segmnt(x+1,y+16,x+12,y+16);
1482         segmnt(x+12,y+36,xx+28,yy-8);
1483         segmnt(xx+28,yy-8,xx+20,yy-8);
1484         lftarrow(xx+20,yy-8);
1485         rohtsemi(x+12,y+26);
1486         break;
1487     case 4:
1488         segmnt(x+1,y-16,x+12,y-16);
1489         segmnt(x+12,y-36,xx+28,yy+8);
1490         segmnt(xx+28,yy+8,xx+20,yy+8);
1491         lftarrow(xx+20,yy+8);
1492         rohtsemi(x+12,y-26);
1493         break;
1494     }
1495 return;
1496 }
1497
1498
1499 rhtsemi(x,y) {
1500     int i,j,k;

```



```

1501     double sqrt();
1502
1503     for (k=0;k<11;k++) {
1504         i = (y-(sqrt(100.-k*k)));
1505         j = (y+(sqrt(100.-k*k)));
1506         dot(x+(k/2.),511.-i);
1507         dot(x+(k/2.),511.-j);
1508         if(k>8) {
1509             dot(x+(k/2.),511.-i-2);
1510             dot(x+(k/2.),511.-i-1);
1511             dot(x+(k/2.),511.-i+1);
1512             dot(x+(k/2.),511.-i+2);
1513             dot(x+(k/2.),511.-j-2);
1514             dot(x+(k/2.),511.-j-1);
1515             dot(x+(k/2.),511.-j+1);
1516             dot(x+(k/2.),511.-j+2);
1517         }
1518     }
1519     return;
1520 }
1521
1522
1523 lftarrow(x,y) {
1524
1525     segmnt(x+4,y-4,x,y);
1526     segmnt(x+4,y+4,x,y);
1527     return;
1528 }
1529
1530
1531 intrns2(xx,yy,x,v) {
1532     int inout;
1533
1534     if(x>(x+12)) {
1535         if(yy<=y) {
1536             if(yy==y) inout=0;
1537             else inout=1;
1538         }
1539         else inout=2;
1540     }
1541     else {
1542         if(yy<=y) inout=4;
1543         else inout=3;
1544     }
1545
1546     switch (inout) {
1547         case 0:
1548             segmnt(xx-16,yy,x+1,y);
1549             lftarrow(x+1,y);
1550             break;
1551         case 1:
1552             segmnt(xx-12,yy+12,x+12,y-8);
1553             segmnt(x+12,y-8,x+1,y-8);
1554             lftarrow(x+1,y-8);
1555             break;
1556         case 2:
1557             segmnt(xx-12,yy-12,x+12,y+8);
1558             segmnt(x+12,y+8,x+1,y+8);
1559             lftarrow(x+1,y+8);
1560             break;

```



```

1561         case 3:
1562             segmnt(xx+12,yy-12,xx+12,yy+36);
1563             segmnt(x+12,y+16,x+1,y+16);
1564             rghtsemi(x+12,y+26);
1565             lftarrow(x+1,y+16);
1566             break;
1567         case 4:
1568             segmnt(xx+12,yy+12,xx+12,yy-36);
1569             segmnt(x+12,y-16,x+1,y-16);
1570             rghtsemi(x+12,y-26);
1571             lftarrow(x+1,y-16);
1572             break;
1573     }
1574     return;
1575 }
1576
1577
1578 outrns2(xx,yy,x,y) {
1579     int output;
1580
1581     if(xx<(x-12)) {
1582         if(yy<=y) {
1583             if(yy==y) output=0;
1584             else output=1;
1585         }
1586         else output=2;
1587     }
1588     else {
1589         if(yy<=y) output=4;
1590         else output=3;
1591     }
1592
1593     switch (output) {
1594         case 0:
1595             segmnt(x-1,y,xx+20,yy);
1596             lftarrow(xx+20,yy);
1597             break;
1598         case 1:
1599             segmnt(x-12,y-8,xx+28,yy+8);
1600             segmnt(xx+28,yy+8,xx+20,yy+8);
1601             lftarrow(xx+20,yy+8);
1602             segmnt(x-1,y-8,x-12,y-8);
1603             break;
1604         case 2:
1605             segmnt(x-12,y+8,xx+28,yy-8);
1606             segmnt(xx+28,yy-8,xx+20,yy-8);
1607             lftarrow(xx+20,yy-8);
1608             segmnt(x-1,y+8,x-12,y+8);
1609             break;
1610         case 3:
1611             segmnt(x-1,y+16,x-12,y+16);
1612             segmnt(x-12,y+36,xx-28,yy-8);
1613             segmnt(xx-28,yy-8,xx-20,yy-8);
1614             rghtarrow(xx-20,yy-8);
1615             lftsemi(x-12,y+26);
1616             break;
1617         case 4:
1618             segmnt(x-1,y-16,x-12,y-16);
1619             segmnt(x-12,y-36,xx-28,yy+8);
1620             segmnt(xx-28,yy+8,xx-20,yy+8);

```



```

1621         rghtarrow(xx-20,yy+8);
1622         lftsemi(x-12,y-26);
1623         break;
1624     }
1625     return;
1626 }
1627
1628
1629 coltab() {
1630     int i;
1631
1632     i=11*16;
1633     lodcol(i++,15,15,15); /* color 0 */
1634     lodcol(i++,0,10,0); /* color 1 */
1635     lodcol(i++,15,0,0); /* color 2 */
1636     lodcol(i++,15,15,0); /* color 3 */
1637     lodcol(i++,12,0,12); /* color 4 */
1638     lodcol(i++,5,5,12); /* color 5 */
1639     lodcol(i++,6,6,5); /* color 6 */
1640     lodcol(i++,5,3,3); /* color 7 */
1641     lodcol(i++,10,0,10); /* color 8 */
1642     lodcol(i++,12,5,5); /* color 9 */
1643     lodcol(i++,5,5,3); /* color 10 */
1644     lodcol(i++,12,12,0); /* color 11 */
1645     lodcol(i++,8,7,3); /* color 12 */
1646     lodcol(i++,5,4,2); /* color 13 */
1647     lodcol(i++,0,0,6); /* color 14 */
1648     lodcol(i++,6,0,0); /* color 15 */
1649     return;
1650 }
1651
1652
1653     /*****
1654     /*****
1655     /***** END OF PROGRAM TRANSGRAPH.C *****/
1656     /*****
1657     /*****
1658

```



```

1 #
2
3 /*****
4 /*****
5 /*****
6 /*****          PROGRAM LINKGRAPH.C          *****/
7 /*****
8 /*****          STEPHEN C. JENNINGS JC91 USMC          *****/
9 /*****          ROBERT J. HARTEL CS91 USA            *****/
10 /*****
11 /*****          WRITTEN FALL QUARTER 1979            *****/
12 /*****          NAVAL POSTGRADUATE SCHOOL            *****/
13 /*****          MONTEREY, CALIFORNIA                *****/
14 /*****
15 /*****
16 /*****
17
18
19 /*****
20 /*****          EXTERNAL DECLARATIONS          *****/
21 /*****
22
23
24 /**** LITERALS ****/
25
26 #define header      4      /* contains control # as to bytes read in ... */
27 #define std         10     /* the standard input read buffer length .... */
28 #define fired       100    /* the max # of transitions fired in 1 frame */
29 #define frames      150    /* indicates the total # of network states .. */
30 #define bounds      360    /* bounds on max # of nodes or transitions .. */
31 #define upper       2000   /* defines iotbl max length ..... */
32 #define limit       3600   /* defines names max length ..... */
33
34
35 /**** STRUCTURES ****/
36
37 struct {                  /* data structure information on net nodes .. */
38
39     int ctrl1;            /* store control char not used in program ... */
40     int nameptr;          /* index to names array ..... */
41     int marker;           /* initial marker state of the network ..... */
42     int xcord;            /* x coordinate of place ..... */
43     int ycord;           /* y coordinate of place ..... */
44     int plot;            /* whether or not place is to be plotted .... */
45     int length;          /* length of name associated with place ..... */
46
47 }file1 [bounds], *hpl;    /* pointer into data structure ..... */
48
49
50 struct {                  /* data structure information on transitions. */
51
52     int ctrl2;            /* store control char not used in program ... */
53     int trnptr;           /* index to names array ..... */
54     int inptr;            /* pointer to inputs for a transition ..... */
55     int outptr;           /* pointer to outputs for a transition ..... */
56     int xcord;            /* x coordinate of transition ..... */
57     int ycord;           /* y coordinate of transition ..... */
58     int trnplot;         /* whether or not transition is to be plotted */
59     int trnlen;          /* length of name associated with transition */
60

```



```

61 }file2 [bounds], *bp2; /* pointer into data structure ..... */
62
63
64 /**** INTEGERS ****/
65
66 int a1,a2,a3,a4; /* global storage for each data structure ... */
67 int buffer[bounds]; /* buffer into which each frame is read ..... */
68 int cntrl[1]; /* variable containing # transitions fired .. */
69 int ctroverflow; /* keep track of nodes overflow status ..... */
70 int dfltcolor; /* a default color for indicating overflows */
71 int fdfbuf; /* file descriptor for RUN--Y files ..... */
72 int fdgbuf; /* file descriptor for RUN--Z files ..... */
73 int firing[bounds]; /* storage into which fired places are read */
74 int ictr; /* counter passed to a function ..... */
75 int ievents; /* number of non-3 displayable nodes ..... */
76 int iflag; /* counter for the interrupt mechanism ..... */
77 int iotbl[upperl]; /* forms input-to-outout relationship ..... */
78 int ktnframe; /* counter for the iterations of the network. */
79 int linktbl[bounds][4]; /* version 1 & 2 screen nodes locations ..... */
80 int nbrplot; /* counter for number of disolayed nodes .... */
81 int nbytes[2]; /* store count fields for data structures ... */
82 int overflowtbl[fired][2]; /* data structure to store overflow locations */
83 int set; /* user selected conrac graphics screen ..... */
84 int tabcount; /* counter for link revert ..... */
85 int tblctr; /* a counter for version 3..reset conditions. */
86 int uniquetbl[fired][4]; /* reset table locations for version 3 ..... */
87 int vers; /* user selected option ..... */
88 int xinstore[fired]; /* coordinates retained for link revert ..... */
89 int xoutstore[fired]; /* coordinates retained for link revert ..... */
90 int yinstore[fired]; /* coordinates retained for link revert ..... */
91 int youtstore[fired]; /* coordinates retained for link revert ..... */
92
93
94 /**** CHARACTERS ****/
95
96 char cbuf[std]; /* buffer store of file to be executed ..... */
97 char dbuf[std]; /* buffer store determining data scan ..... */
98 char fbuf[std]; /* buffer to store name of second file ..... */
99 char gbuf[std]; /* buffer to store name of third file..... */
100 char names[limit]; /* character array for node labels ..... */
101 char scrn; /* option variable for display to the screen. */
102 char tbuf[std]; /* buffer to store timing variable ..... */
103 char vbuf[std]; /* buffer to store version selected ..... */
104 char timing; /* variable to set program execution timing */
105
106
107 /***** */
108 /**** FUNCTION MAIN ****/
109 /***** */
110
111
112 main() {
113
114     extern rubout(); /* declare 'rubout' globally ..... */
115     init(); /* read input file ..... */
116     determine(); /* verify if user wants to see data structure */
117     display(); /* display input to crt ..... */
118     select(); /* select version of simulation & genisco set */
119     prepare(2); /* prepare genesco-conrac ..... */
120     drawnode(); /* draw network nodes on conrac ..... */

```



```

121     places();          /* verify correct nodes drawn ..... */
122     link();            /* function depicts network connectivity .... */
123     imark();           /* starting status of network packets ..... */
124     signal(2,rubout);  /* sets 'BRK' as interrupt ..... */
125     marking();        /* successive iterations of network flow .... */
126     gnfini();         /* closing out graphics facilities ..... */
127
128 }
129
130
131 /*****
132  *****/
133 /*****
134
135
136 pause(period) {
137     /* function necessary as sleep() not compatible with signal() */
138     int i,j,k;
139
140     printf("***>>interrupt.....");
141     for(i=0;i<period;i++) {
142         for(j=0;j<400;j++) {
143             for(k=0;k<1000;k++) {
144                 ;
145             }
146         }
147         printf("***>>wait.....");
148     }
149     return;
150 }
151
152 rubout() {
153     /* function enables the 'brk' key as the interrupt signal */
154     char halt;
155
156     space(2);
157     printf
158     ("***>> received signal...frame number %d...<ret> to continue \n",
159     (iflag+2));
160     printf
161     ("***>> for termination of program...type 'brk' from console \n");
162     while ((halt=getchar())!='\n') {
163         /* do-nothing loop */
164     }
165     signal(2,rubout);
166     return;
167 }
168
169
170 space(returns) {
171     int i;
172
173     for (i=0; i<returns; i++) {
174         printf("\n");
175     }
176     return;
177 }
178
179
180 init() {

```



```

181      /* function opens unformatted file & initializes start condition */
182      int a,bufctr,count,fd,i,j;
183      char c,f;
184
185      space(2);
186      printf("***->LINKGRAPH ILLUSTRATES NETWORK SIMULATION MODELS");
187      space(2);
188      printf("***->ENTER THE NAME OF THE FILE TO BE PROCESSED..BUT \n");
189      printf("      NOTE THAT THIS FILE MUST BE AN UNFORMATTED FILE \n");
190      printf("      PRODUCED AS A RESULT OF EXECUTING simulator.out \n");
191      printf("      THE LAST LETTER OF WHICH MUST END IN LETTER 'X' \n");
192  error:space(2);
193      printf("***->");
194
195      i=0;
196      while((c=getchar()) != '\n') {
197          cbuf[i]=c;
198          i++;
199      }
200      cbuf[i]='\0';
201
202      bufctr=i;
203      for(j=0;j<bufctr;j++) {
204          qbuf[j]=fbuf[j]=cbuf[j];
205          if(cbuf[j]=='X') {
206              fbuf[j]='Y';
207              abuf[j]='Z';
208              qbuf[j+1]=fbuf[j+1]='\0';
209              j=bufctr;
210          }
211      }
212
213      fd = open(cbuf,0);
214      if (fd <= 0) {
215          printf
216          ("***->error occurred in opening file.....try again");
217          space(3);
218          goto error;
219      }
220
221      if((count = read(fd,nbytes,header)) != header)
222          printf("error occurred in nbytes read");
223      ievents = (nbytes[1] - 1);
224      a1 = nbytes[1];
225      a = (nbytes[1]-1)*14;
226      if((count = read(fd,file1,a)) != a)
227          printf("error occured in file1 read");
228
229      if((count=read(fd,nbytes,header))!=header)
230          printf("error occurred in nbytes read");
231      a2=nbytes[1];
232      a=(nbytes[1]-1)*16;
233      if((count=read(fd,file2,a))!=a)
234          printf("error occurred in file2 read");
235
236      if((count=read(fd,nbytes,header))!=header)
237          printf("error occurred in header read");
238      a3=nbytes[1];
239      a=(nbytes[1]+1)*2;
240      if((count=read(fd,iotbl,a))!=a)

```



```

241     printf("error occured in iotbl read");
242
243     if((count=read(fd,nbytes,header))!=header)
244         printf("error occurred in header read");
245     a4=nbytes[1];
246     a=nbytes[1]+1;
247     if((count=read(fd,names,a))!=a)
248         printf("error occurred in names read");
249
250     close(fd);
251     return;
252 }
253
254
255 determine() {
256     int i;
257     char d;
258
259     space(2);
260     printf("***->FUNCTION 'DETERMINE' ALLOWS THE USER \n");
261     printf("    TO EXAMINE ALL PRIMARY DATA STRUCTURES");
262     over:space(2);
263     printf
264     ("***->IF THIS FEATURE IS DESIRED TYPE 1 IF NOT 0, THEN <RET>");
265     space(2);
266     printf("***->");
267
268     i=0;
269     while((d=getchar())!='\n') {
270         dbuf[i]=d;
271         i++;
272     }
273     dbuf[i]='\0';
274
275     i=0;
276     while(dbuf[i]!='\0') {
277         d=dbuf[i];
278         switch(d) {
279             case '0':
280                 scrn='0';
281                 break;
282             case '1':
283                 scrn='1';
284                 printf("***->USE CONTROL Q WHEN SCREEN FULL");
285                 break;
286             default:
287                 printf("***->either blank or invalid entry");
288                 goto over;
289                 break;
290         }
291         i++;
292     }
293     return;
294 }
295
296
297 display() {
298     int i;
299
300     if(scrn=='1') {

```



```

301     space(2);
302     bp1 = file1;
303     printf("***-> FILE1 DATA STRUCTURE");
304     space(2);
305     printf
306     ("Infeed  #   marker   xcord   ycord   plot   length \n");
307     for (i=0;i<a1; i++){
308         printf("%d \t %d \t %d \t %d \t %d \t %d \t %d \t \n",
309             bp1->ctrl1, bp1->nameptr, bp1->marker, bp1->xcord,
310             bp1->ycord, bp1->plot, bp1->length);
311         bp1++;
312     }
313
314     space(2);
315     bp2 = file2;
316     printf("***-> FILE2 DATA STRUCTURE");
317     space(2);
318     printf
319     ("Infeed  trnptr  intrn   outtrn  xxcord  yycord  trnplot  trnlen \n");
320     for (i=0;i<a2; i++){
321         printf("%d \t %d \t %d \t %d \t %d \t %d \t %d \t \n",
322             bp2->ctrl2, bp2->trnptr, bp2->intrn, bp2->outtrn,
323             bp2->xxcord, bp2->yycord, bp2->trnplot, bp2->trnlen);
324         bp2++;
325     }
326
327     space(2);
328     printf("***-> IOTBL DATA ARRAY");
329     space(2);
330     for (i=0; i<a3; i++) {
331         printf("%d ", iotbl[i]);
332     }
333
334     space(2);
335     printf("***-> NAMES DATA ARPAY");
336     space(2);
337     for (i=0; i<a4+1; i++) {
338         printf("%c", names[i]);
339     }
340     space(2);
341 }
342 return;
343 }
344
345
346 prepare(type) {
347     /* function designates set, screen size and color table */
348     int n,t,v;
349
350     y=0;
351     denisco (set);
352     erase();
353     screen(0.0,0.0,511.0,511.0);
354     setmod(type);
355     coltan();
356     colort(11);
357     for(n=12;n<14;n++) {
358         color(n);
359         for(t=0;t<512;t++) {
360             segmnt(0,v+t,511,y+t);

```



```

361     }
362 }
363 return;
364 }
365
366
367 drawnode() {
368     /* function displays type & location of network nodes */
369     char c,*nptr,hold;
370     int a,b,clrlbl,count,d,entry,h,i,j,k,l,node,test,x,y,*z;
371     float s;
372     double sqrt();
373
374     bol = file1;
375     a=0;
376     count = 1;
377
378     while((test=(bol->nameptr))!=0) {
379         color(14);
380         test++;
381         z = &names[test];
382         c = *z;
383
384         if ((b=(bol->plot))!= 0) {
385             switch (c){
386                 case 'I':
387                     node=1;
388                     x = (bol->xcord);
389                     y = (bol->ycord);
390                     for(d=0;d<10;d++) {
391                         segmnt(x-16,y+2+d,x,y+2+d);
392                     }
393                     if(vers==3) {
394                         nptr = &names[test+1];
395                         hold = *nptr;
396                         clrlbl=atoi(&hold);
397                         label(x,y,test,clrlbl,node);
398                         color(14);
399                     }
400                     else {
401                         clrlbl=14;
402                         label(x,y,test,clrlbl,node);
403                     }
404                     linktbl[a][3]=1;
405                     break;
406                 case 'O':
407                     node=2;
408                     x = (bol->xcord);
409                     y = (bol->ycord);
410                     for(d=0;d<10;d++) {
411                         segmnt(x-16,y-2-d,x,y-2-d);
412                     }
413                     if(vers==3) {
414                         nptr = &names[test+1];
415                         hold = *nptr;
416                         clrlbl=atoi(&hold);
417                         label(x,y,test,clrlbl,node);
418                         color(14);
419                     }
420                     else {

```



```

421             clrbl=14;
422             label(x,y,test,clrbl,node);
423         }
424         linktbl[al][3]=1;
425         break;
426     case 'R':
427         node=0;
428         x = (bpl->xcord);
429         y = (bpl->ycord);
430         for(d=0;d<31;d++) {
431             segmnt
432             (x-18+d/4,y+15-d,x+18-d/4,y+15-d);
433         }
434         clrbl=14;
435         label(x,y,test,clrbl,node);
436         break;
437     case 'S':
438         node=0;
439         x = (bpl->xcord);
440         y = (bpl->ycord);
441         for(d=0;d<31;d++) {
442             segmnt(x-18,y-15+d,x+18,y-15+d);
443         }
444         if(vers==3) {
445             nptr = &names[test+1];
446             hold = *nptr;
447             clrbl=atoi(&hold);
448             label(x,y,test,clrbl,node);
449             color(14);
450         }
451         else {
452             clrbl=14;
453             label(x,y,test,clrbl,node);
454         }
455         break;
456     case 'I':
457         node=0;
458         x = (bpl->xcord);
459         y = (bpl->ycord);
460         for (k=0;k<19;k++) {
461             s=k;
462             i=(x-(sqrt(324.-s*s)));
463             j=y+s;
464             l=(x+(sqrt(324.-s*s)));
465             segmnt(i,j,l,j);
466         }
467         for (k=0;k<19;k++) {
468             s=k;
469             i=(x-(sqrt(324.-s*s)));
470             j=y-s;
471             l=(x+(sqrt(324.-s*s)));
472             segmnt(i,j,l,j);
473         }
474         if(vers==3) {
475             nptr = &names[test+1];
476             hold = *nptr;
477             clrbl=atoi(&hold);
478             label(x,y,test,clrbl,node);
479             color(14);
480         }

```



```

481             else {
482                 clrhl=14;
483                 label(x,v,test,clrhl,node);
484             }
485             break;
486         default:
487             printf
488             ("name not valid identifier");
489             space(2);
490             break;
491     }
492
493     linktbl[a][0]=count;
494     linktbl[a][1] = x;
495     linktbl[a][2] = y;
496     a++;
497 }
498 count++;
499 bpl++;
500 }
501 if(vers==3) {
502     entry=0;
503     uniquetbl[entry][0]=linktbl[0][0];
504     uniquetbl[entry][1]=linktbl[0][1];
505     uniquetbl[entry][2]=linktbl[0][2];
506     uniquetbl[entry][3]=linktbl[0][3];
507     tblctr=1;
508     for(i=0;i<a;i++) {
509         if(uniquetbl[entry][1]==linktbl[i+1][1] &&
510            uniquetbl[entry][2]==linktbl[i+1][2]) {
511             /* do nothing */
512         }
513         else {
514             entry++;
515             uniquetbl[entry][0]=linktbl[i+1][0];
516             uniquetbl[entry][1]=linktbl[i+1][1];
517             uniquetbl[entry][2]=linktbl[i+1][2];
518             uniquetbl[entry][3]=linktbl[i+1][3];
519             tblctr++;
520         }
521     }
522 }
523 nrcrlot=a;
524 return;
525 }
526
527
528 places() {
529     int h,i,j,k;
530
531     if(scrn=='1') {
532         space(2);
533         printf("++->DATA STRUCTURE LINKTBL ");
534         space(2);
535         for(i=0;i<nrcrlot;i++) {
536             for(h=0;h<4;h++) {
537                 printf("%d --",linktbl[i][h]);
538             }
539             space(1);
540         }

```



```

541     space(2);
542     if(vers==3) {
543         printf("***->DATA STRUCTURE UNIQUEtbl");
544         space(2);
545         for(j=0;j<tblctr;j++) {
546             for(k=0;k<4;k++) {
547                 printf("%d --",uniquetbl[j][k]);
548             }
549             space(1);
550         }
551     }
552     space(2);
553 }
554 return;
555 }
556
557
558 label(xx,yy,zz,clbl,place) {
559     /* determines node label placement in relation to node */
560     int f,i;
561
562     color(clbl);
563
564     switch (place) {
565     case 0 :
566         if(xx>250) {
567             for(i=0;i<2;i++) {
568                 charac((xx+(22+(8*i))),yy,names[zz]);
569                 zz++;
570             }
571         }
572         else {
573             for(i=0;i<2;i++) {
574                 charac((xx-(34-(8*i))),yy,names[zz]);
575                 zz++;
576             }
577         }
578         break;
579     case 1 :
580         if(xx>250) {
581             for(i=0;i<2;i++) {
582                 charac((xx+4+(8*i)),yy+2,names[zz]);
583                 zz++;
584             }
585         }
586         else {
587             for(i=0;i<2;i++) {
588                 charac((xx-(32-(8*i))),yy+2,names[zz]);
589                 zz++;
590             }
591         }
592         break;
593     case 2 :
594         if(xx>250) {
595             for(i=0;i<2;i++) {
596                 charac((xx+4+(8*i)),yy-10,names[zz]);
597                 zz++;
598             }
599         }
600         else {

```



```

601         for(i=0;i<2;i++) {
602             charac((xx-(32-(8*i))),yy-10,names[zz]);
603             zz++;
604         }
605     }
606     break;
607 }
608 return;
609 }
610
611
612 link() {
613     /* function links nodes by information stored in iotbl */
614     int a,b,in[40],out[40],plotin[40],plotout[40];
615     int content,ktr,i,k,m,maxinctr,maxoutctr,l,p,inx,iny,outx,outy;
616
617     color(14);
618     content=iotbl[1];
619     ktr=1;
620     while(content!=0) {
621         maxinctr=iotbl[ktr];
622         for(i=0;i<content;i++) {
623             ktr++;
624             in[i]=iotbl[ktr];
625         }
626         ktr++;
627         content=iotbl[ktr];
628         maxoutctr=iotbl[ktr];
629         for(i=0;i<content;i++) {
630             ktr++;
631             out[i]=iotbl[ktr];
632         }
633         for(n=0;n<maxinctr;n++) {
634             plotin[n]=0;
635             for(l=0;l<nbrplot;l++) {
636                 if(in[n]==linktbl[l][0]) {
637                     plotin[n]=1;
638                     l=nbrplot;
639                 }
640             }
641         }
642         for(p=0;p<maxoutctr;p++) {
643             plotout[p]=0;
644             for(l=0;l<nbrplot;l++) {
645                 if(out[p]==linktbl[l][0]) {
646                     plotout[p]=1;
647                     l=nbrplot;
648                 }
649             }
650         }
651     }
652
653     for(k=0;k<maxinctr;k++) {
654         for(i=0;i<maxoutctr;i++) {
655             if(plotin[k]==1 && plotout[i]==1) {
656                 a=in[k];
657                 b=out[i];
658                 for(l=0;l<nbrplot;l++) {
659                     if(linktbl[l][0]==a) {
660                         inx=linktbl[l][1];

```



```

661             iny=linktbl[1][2];
662             l=nbplot;
663         }
664     }
665     for(m=0;m<nbplot;m++) {
666         if(linktbl[m][0]==b) {
667             outx=linktbl[m][1];
668             outy=linktbl[m][2];
669             m=nbplot;
670         }
671     }
672     lines(inx,iny,outx,outy);
673 }
674 }
675 }
676 ktr++;
677 content=iotbl[ktr];
678 }
679 return;
680 }
681
682
683 lines(x1,v1,x2,v2) {
684     int lnkcase,x,y;
685
686     if(x1<=x2) {
687         if(y1<=v2) {
688             if(x1==x2) lnkcase=3;
689             else {
690                 if(y1==v2) lnkcase=0;
691                 else lnkcase=4;
692             }
693         }
694         else {
695             if(x1==x2) lnkcase=1;
696             else lnkcase=5;
697         }
698     }
699     else {
700         if(y1<=v2) {
701             if(v1==v2) lnkcase=2;
702             else lnkcase=7;
703         }
704         else lnkcase=6;
705     }
706     switch(lnkcase) {
707         case 0:
708             segtnt(x1+20,v1,x2-20,v2);
709             x=x2-20;
710             v=v2;
711             segtnt(x-4,v-4,x,y);
712             segtnt(x-4,v+4,x,y);
713             break;
714         case 1:
715             segtnt(x1,y1-20,x2,v2+20);
716             x=x2;
717             v=v2+20;
718             segtnt(x-4,v+4,x,y);
719             segtnt(x+4,v+4,x,y);
720             break;

```



```

721     case 2:
722         segmnt(x1-20,y1,x2+20,y2);
723         x=x2+20;
724         y=y2;
725         segmnt(x+4,y-4,x,y);
726         segmnt(x+4,y+4,x,y);
727         break;
728     case 3:
729         segmnt(x1,y1+20,x2,y2-20);
730         x=x2;
731         y=y2-20;
732         segmnt(x-4,y-4,x,y);
733         segmnt(x+4,y-4,x,y);
734         break;
735     case 4:
736         segmnt(x1+24,y1+24,x2-24,y2-24);
737         x=x2-16;
738         y=y2-16;
739         segmnt(x,y-6,x,y);
740         segmnt(x-6,y,x,y);
741         segmnt(x2-24,y2-24,x,y);
742         break;
743     case 5:
744         segmnt(x1+24,y1-24,x2-24,y2+24);
745         x=x2-16;
746         y=y2+16;
747         segmnt(x,y+6,x,y);
748         segmnt(x-6,y,x,y);
749         segmnt(x2-24,y2+24,x,y);
750         break;
751     case 6:
752         segmnt(x1-24,y1-24,x2+24,y2+24);
753         x=x2+16;
754         y=y2+16;
755         segmnt(x,y+6,x,y);
756         segmnt(x+6,y,x,y);
757         segmnt(x2+24,y2+24,x,y);
758         break;
759     case 7:
760         segmnt(x1-24,y1+24,x2+24,y2-24);
761         x=x2+16;
762         y=y2-16;
763         segmnt(x,y-6,x,y);
764         segmnt(x+6,y,x,y);
765         segmnt(x2+24,y2-24,x,y);
766         break;
767 }
768 return;
769 }
770
771
772 select() {
773     int i,n;
774     char v;
775
776     space(2);
777     printf("***-->THERE ARE 3 VERSIONS TO THIS GRAPHICS PACKAGE \n");
778     printf("PLEASE SELECT ONE OF THE FOLLOWING VERSIONS: \n");
779     again:space(2);
780     printf("VERSION 1 ... PETRI-NET PACKAGE ..... TYPE 1 \n");

```



```

781     printf("        VERSION 2 ... PACKET REPRESENTATION ... TYPE 2 \n");
782     printf("        VERSION 3 ... MULTIROUTING PACKAGE .... TYPE 3 \n");
783     n = 0;
784     twice:space(2);
785     printf("***->");
786     n++;
787     i = 0;
788     while ((v=getchar()) != '\n') {
789         vbuf[i] = v;
790         i++;
791     }
792     vbuf[i] = '\0';
793
794     i = 0;
795     while(vbuf[i] != '\0') {
796         v = vbuf[i];
797         if(n==1) {
798             switch(v) {
799                 case '1':
800                     vers = 1;
801                     break;
802                 case '2':
803                     vers = 2;
804                     break;
805                 case '3':
806                     vers = 3;
807                     break;
808                 default:
809                     printf
810                     ("***->incorrect version try again!");
811                     goto again;
812                     break;
813             }
814             i++;
815         }
816         else {
817             switch(v) {
818                 case '0':
819                     set = 0;
820                     break;
821                 case '1':
822                     set = 1;
823                     break;
824                 case '2':
825                     set = 2;
826                     break;
827                 default:
828                     printf
829                     ("***->incorrect genisco set try again");
830                     printf
831                     ("      set selection should be 0,1,or2");
832                     n = 1;
833                     goto twice;
834                     break;
835             }
836             i++;
837         }
838     }
839     if(n==1) {
840         space(2);

```



```

841         printf("***->NOW SELECT THE GENISCO SET YOU WISH \n");
842         printf("        THE PROGRAM TO BE DISPLAYED TO..... \n");
843         printf("        IN C3 LAB EITHER SET0, SET1 OR SET2 \n");
844         goto twice;
845     }
846     return;
847 }
848
849
850 imark() {
851     /* marks initial state of system by calling appropriate function */
852     int o,colour,e,g,x,y;
853
854     bpl = file1;
855     dfltcolor=3;
856     color(dfltcolor);
857     colour=2;
858     ctroverflow=0;
859     while ((g=(bpl->nameptr)) !=0) {
860         if((b=(bpl->plot)) !=0) {
861             switch(vers) {
862                 case 1:
863                     ivers1();
864                     break;
865                 case 2:
866                     ivers2(colour);
867                     break;
868                 case 3:
869                     ivers3(colour);
870                     break;
871             }
872         }
873         if(dfltcolor!=3) color(3);
874         bpl++;
875     }
876     color(14);
877     printf(0,350.,480., "TIME FRAME = 1");
878     preread(1);
879     hilite ();
880     displa();
881     hold();
882     reset(1);
883     color(13);
884     overflow();
885     printf(0,350.,480., "TIME FRAME = 1");
886     return;
887 }
888
889
890 hold() {
891     int i;
892
893     time:space(2);
894     printf("***->THIS IS THE INITIAL STATE OF THE NETWORK \n");
895     printf("        ENTER THE TIMING MODE FOR EXECUTION .... \n");
896     printf("        '0' FOR NO DELAYS ..'1' FOR FRAME PAUSES \n");
897     printf("***->");
898     i = 0;
899     while((timing=getchar()) != '\n') {
900         tbuf[i] = timing;

```



```

901         i++;
902     }
903     tbuf[i] = '\0';
904
905     i = 0;
906     while(tbuf[i] != '\0') {
907         timind = tbuf[i];
908         i++;
909     }
910     if(timind!='0' && timind!='1') {
911         printf("***->incorrect version try again!");
912         goto time;
913     }
914     return;
915 }
916
917
918 ivers1() {
919     int e,x,y,z;
920     char check;
921
922     e=(bol->marker);
923     x=(bol->xcord);
924     y=(bol->ycord);
925     z=(bol->nameptr);
926     if((check=names[z+1])!='I' && (check=names[z+1])!='0') {
927         printf(0,x-3.0,511.-(y-3),"%d",e);
928     }
929     else {
930         if(check=names[z+1]=='I') printf(0,x-14.0,511.-(y+2),"%d",e);
931         else printf(0,x-14.0,511.-(y-9),"%d",e);
932     }
933     return;
934 }
935
936
937 ivers2(colour) {
938     int e,x,y,z;
939     char check;
940
941     e=(bol->marker);
942     x=(bol->xcord);
943     y=(bol->ycord);
944     z=(bol->nameptr);
945     if((check=names[z+1])!='I' && (check=names[z+1])!='0') {
946         ockt2(x,y,e,colour);
947     }
948     else {
949         if(check=names[z+1]=='I') printf(0,x-14.0,511.-(y+2),"%d",e);
950         else printf(0,x-14.0,511.-(y-9),"%d",e);
951     }
952     return;
953 }
954
955
956 ivers3(colour) {
957     char keep,*kptr;
958     int a,aa,b,bb,c,cc,d,rfired,i,k,n,stack,total,x,y;
959
960     total=0;

```



```

961     n=1;
962     x=(bol->xcord);
963     y=(bol->ycord);
964     a=(bol->length);
965     b=(bol->nameptr);
966     c=(bol->marker);
967     total=total+c;
968
969     if(c>0) {
970         for(i=0;i<c;i++) {
971             kptr = &names[b+a-2];
972             keep = *kptr;
973             clr[n]=atoi(&keep);
974             n++;
975         }
976     }
977
978     kptr = &names[b+a];
979     keep = *kptr;
980     stack=atoi(&keep);
981
982     if(names[b+(a-1)]!='0') {
983         if(names[n+(a-1)]=='2') stack=stack+20;
984         else stack = stack + 10;
985     }
986
987     for(i=0;i<stack-1;i++) {
988         bol++;
989         aa=(bol->length);
990         bb=(bol->nameptr);
991         cc=(bol->marker);
992         total=total+cc;
993         if(cc>0) {
994             for(k=0;k<cc;k++) {
995                 kptr = &names[bb+aa-2];
996                 keep = *kptr;
997                 clr[n]=atoi(&keep);
998                 n++;
999             }
1000         }
1001     }
1002     if(names[n+1]!='I' && names[b+1]!='0') {
1003         pkt3
1004         (x,v,total,clr[1],clr[2],clr[3],clr[4],clr[5],clr[6],clr[7],
1005         colour);
1006     }
1007     else {
1008         if(names[n+1]=='I') printf(0,x-14.0,511.-(y+2),"%d",total);
1009         else printf(0,x-14.0,511.-(y-9),"%d",total);
1010     }
1011     return;
1012 }
1013
1014
1015 preread(flag) {
1016     int bucket[2],count,fd,fd,i,nbrtrns;
1017     if(flag!=3) {
1018         if(flag==1) {
1019             fd = open (fbuf, 0);
1020             if (fd<=0) {

```



```

1021         printf("***->error occurred in opening fd file");
1022     }
1023     fdfbuf=fd;
1024     fg = open (qbuf, 0);
1025     if (fg<=0) {
1026         printf("***->error occurred in opening fg file");
1027     }
1028     fdgbuf=fg;
1029 }
1030 if((count=read (fdfbuf, bucket, 2))!=2) {
1031     printf("***->error occurred in fd bucket read");
1032 }
1033 if((count=read (fdfbuf, buffer,(ievents*2)))!=(ievents*2)) {
1034     printf("***->error occurred in buffer read");
1035 }
1036 if((count=read (fdgbuf, bucket, 2))!=2) {
1037     printf("***->error occurred in fg bucket read");
1038 }
1039 if((count=read (fdgbuf, cntnl,2))!=2) {
1040     printf("***->error occurred in cntnl read");
1041 }
1042 if(cntnl[0]!=0) {
1043     space(2);
1044     printf("***->the last network state has been achieved");
1045     kthframe=frames+1;
1046     space(2);
1047 }
1048 else {
1049     if((count=read (fdgbuf, bucket, 2))!=2) {
1050         printf("***->error occurred in bucket read");
1051     }
1052     nbrtrns = cntnl[0]*2;
1053     if((count=read (fdgbuf, firing,nbrtrns))!=nbrtrns) {
1054         printf("***->error occurred in firing read");
1055     }
1056 }
1057 }
1058 else {
1059     close(fdfbuf);
1060     close(fdgbuf);
1061 }
1062 return;
1063 }
1064
1065
1066 hilite() {
1067     int q,i,in[25],intbl,j,k,l,m,maxinctr,maxoutctr,n,out[25],
1068         outtbl,p,oletin[25],oletout[25],inx,iny,qutx,outy;
1069
1070     tabcount=0;
1071     for(i=0;i<cntnl[0];i++) {
1072         bp2=file2;
1073         for(j=0;i<firing[i]-1;j++) {
1074             bp2++;
1075         }
1076         intbl=(bp2->intcn);
1077         outtbl=(bp2->outtrn);
1078         maxinctr=jtbl[intbl];
1079         for(k=0;k<maxinctr;k++) {
1080             intbl++;

```



```

1081         in[k]=iotbl(intbl);
1082     }
1083     maxoutctr=iotbl(outtbl);
1084     for(l=0;l<maxoutctr;l++) {
1085         outtbl++;
1086         out[l]=iotbl(outtbl);
1087     }
1088     for(p=0;p<maxinctr;p++) {
1089         plotin[p]=0;
1090         for(l=0;l<nbrplot;l++) {
1091             if(in[p]==linktbl[l][0]) {
1092                 plotin[p]=1;
1093                 l=nbrplot;
1094             }
1095         }
1096     }
1097
1098     for(p=0;p<maxoutctr;p++) {
1099         plotout[p]=0;
1100         for(l=0;l<nbrplot;l++) {
1101             if(out[p]==linktbl[l][0]) {
1102                 plotout[p]=1;
1103                 l=nbrplot;
1104             }
1105         }
1106     }
1107     for(k=0;k<maxinctr;k++) {
1108         for(q=0;q<maxoutctr;q++) {
1109             if(plotin[k]==1 && plotout[q]==1) {
1110                 for(m=0;m<nbrplot;m++) {
1111                     if(linktbl[m][0]==in[k]) {
1112                         inx=linktbl[m][1];
1113                         iny=linktbl[m][2];
1114                         m=nbrplot;
1115                     }
1116                 }
1117                 for(n=0;n<nbrplot;n++) {
1118                     if(linktbl[n][0]==out[q]){
1119                         outx=linktbl[n][1];
1120                         outy=linktbl[n][2];
1121                         n=nbrplot;
1122                     }
1123                 }
1124                 color(11);
1125                 lines(inx,iny,outx,outy);
1126                 xinstore[tandcount]=inx;
1127                 yinstore[tandcount]=iny;
1128                 xoutstore[tandcount]=outx;
1129                 youtstore[tandcount]=outy;
1130                 tandcount++;
1131             }
1132         }
1133     }
1134 }
1135 return;
1136 }
1137
1138
1139 linkvt() {
1140     int i,inx,iny,outx,outy;

```



```

1141
1142     color(14);
1143     for(i=0;i<stabcount;i++) {
1144         inx= xinstore[i];
1145         iny= yinstore[i];
1146         outx= xoutstore[i];
1147         outy= youtstore[i];
1148         lines(inx,iny,outx,outy);
1149     }
1150 return;
1151 }
1152
1153
1154 stage() {
1155
1156     if (kthframe!=0 && kthframe<frames+1) {
1157         oreread(2);
1158         if(kthframe!=frames+1) {
1159             hilite();
1160             if(timing=='1') {
1161                 disola();
1162                 pause(1);
1163             }
1164         }
1165     }
1166 return;
1167 }
1168
1169
1170 marking() {
1171     /* function displays successive iterations of the network */
1172     int colour,draw,i,mark,n,x,y;
1173
1174     bol = file1;
1175     n = 2;
1176
1177     /* following loop processes ievent # data segments each pass */
1178     for(kthframe=0;kthframe<frames;kthframe++) {
1179         stage();
1180         if(kthframe>0) {
1181             reset();
1182             color(13);
1183             overflow();
1184             printf(0,350.,490., "TIME FRAME = %d",n);
1185             n++;
1186         }
1187         iflac = xthframe;
1188         draw = (rac1->plot);
1189         ifltcolor=3;
1190         color (ifltcolor);
1191         colour=2;
1192         ctroverflow=0;
1193         for (i=0; i < ievents; i++) {
1194             if (draw == 1) {
1195                 ictr = i;
1196                 switch (vers) {
1197                     case 1:
1198                         vers1();
1199                         break;
1200                     case 2:

```



```

1201             vers2(colour);
1202             break;
1203         case 3:
1204             vers3(colour);
1205             break;
1206     }
1207 }
1208     i = ictr++;
1209     if(dfltcolor!=3) color(3);
1210     bpl++;
1211     draw = (bpl -> plot);
1212 }
1213     dfltcolor=14;
1214     color(dfltcolor);
1215     colour=13;
1216     printf(0,350.,480., "TIME FRAME = %d",n);
1217     if(timino=='I') pause(2);
1218     lnkrvt();
1219     bpl = file1;
1220 }
1221     preread(3);
1222     return;
1223 }
1224
1225
1226 overflow() {
1227
1228     int i,x,y;
1229
1230     i=0;
1231     while(overflowtbl[i][0]!=0) {
1232         x=overflowtbl[i][0];
1233         y=overflowtbl[i][1];
1234         if(x>250) {
1235             block((x+34)*1.,511.-(y+10)*1.,(x+54)*1.,511.-y*1.);
1236         }
1237         else {
1238             block((x-50)*1.,511.-(y+10)*1.,(x-35)*1.,511.-y*1.);
1239         }
1240         overflowtbl[i][0]=0;
1241         overflowtbl[i][1]=0;
1242         i++;
1243     }
1244     return;
1245 }
1246
1247
1248 vers1() {
1249     int e,x,v,z;
1250     char check;
1251
1252     e=buffer[ictr];
1253     x=(bpl->xcpr);
1254     y=(bpl->ycpr);
1255     z=(bpl->nameptr);
1256     if((check=names[z+1]!='I' && (check=names[z+1]!='O')) {
1257         printf(0,x-3.0,511.-(y-3),"%d",e);
1258     }
1259     else {
1260         if(check=names[z+1]=='I') printf(0,x-14.0,511.-(y+2),"%d",e);

```



```

1261         else printf(0,x-14.0,511.-(y-9),"%d",e);
1262     )
1263 return;
1264 }
1265
1266
1267 vers2(colour) {
1268     int mark,x,y,z;
1269     char check;
1270
1271     x=(bp1->xcord);
1272     y=(bp1->ycord);
1273     z=(bp1->nameptr);
1274     mark=buffer[ictr];
1275     if((check=names[z+1])!='I' && (check=names[z+1])!='0') {
1276         pkt2(x,y,mark,colour);
1277     }
1278     else {
1279         if((check=names[z+1])=='I') printf(0,x-14.0,511.-(y+2),"%d",mark);
1280         else printf(0,x-14.0,511.-(y-9),"%d",mark);
1281     }
1282 return;
1283 }
1284
1285
1286 vers3(colour) {
1287     int a,aa,b,nb,mark,marks,clr[fired],j,k,n,stack,total,x,y;
1288     char keep,*kptr;
1289
1290     total=0;
1291     n=1;
1292     x=(bp1->xcord);
1293     y=(bp1->ycord);
1294     a=(bp1->length);
1295     b=(bp1->nameptr);
1296     mark=buffer[ictr];
1297     total=total+mark;
1298     if(mark>0) {
1299         for(j=0;j<mark;j++) {
1300             kptr = &names[b+a-2];
1301             keep = *kptr;
1302             clr[n]=atoi(&keep);
1303             n++;
1304         }
1305     }
1306
1307     kptr = &names[b+a];
1308     keep = *kptr;
1309     stack=atoi(&keep);
1310
1311     if(names[b+(a-1)]!='0') {
1312         if(names[b+(a-1)]=='1') stack=stack+10;
1313         else {
1314             if(names[b+(a-1)]=='2') stack=stack+20;
1315             else stack = stack + 30;
1316         }
1317     }
1318     for(j=0;j<stack-1;j++) {
1319         bp1++;
1320         ictr++;

```



```

1321     aa=(bp1->length);
1322     bp=(bp1->nameptr);
1323     marks=buffer[ictr];
1324     total=total+marks;
1325     if(marks>0) {
1326         for(k=0;k<marks;k++) {
1327             kptr = &names[bb+aa-2];
1328             keep = *kptr;
1329             clr[n]=atoi(&keep);
1330             n++;
1331         }
1332     }
1333 }
1334 if(names[b+1]!='I' && names[b+1]!='0') {
1335     pkt3
1336     (x,y,total,clr[1],clr[2],clr[3],clr[4],clr[5],clr[6],clr[7],
1337     colour);
1338 }
1339 else {
1340     color(3);
1341     if(names[b+1]=='I') printf(0,x-14.0,511.-(y+2),"%d",total);
1342     else printf(0,x-14.0,511.-(y-9),"%d",total);
1343 }
1344 return;
1345 }
1346
1347
1348 pkt2(xaxis,yaxis,point,class) {
1349
1350     switch(point) {
1351         case 0:
1352             break;
1353         case 1:
1354             block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1355             break;
1356         case 2:
1357             block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1358             block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1359             break;
1360         case 3:
1361             block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1362             block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1363             block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1364             break;
1365         case 4:
1366             block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1367             block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1368             block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1369             block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1370             break;
1371         case 5:
1372             block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1373             block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1374             block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1375             block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1376             block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
1377             break;
1378         case n:
1379             block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1380             block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);

```



```

1381     block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1382     block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1383     block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
1384     block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
1385     break;
1386 case 7:
1387     block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1388     block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1389     block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1390     block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1391     block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
1392     block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
1393     block((xaxis+7)*1.,511.-(yaxis+13)*1.,(xaxis+13)*1.,511.-(yaxis+7)*1.);
1394     break;
1395 default:
1396     overflowtbl[ctroverflow][0]=xaxis;
1397     overflowtbl[ctroverflow][1]=yaxis;
1398     ctroverflow++;
1399     block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1400     block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1401     block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1402     block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1403     block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
1404     block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
1405     block((xaxis+7)*1.,511.-(yaxis+13)*1.,(xaxis+13)*1.,511.-(yaxis+7)*1.);
1406     color(class);
1407     dfltcolor=class;
1408     if(xaxis>250) {
1409         printf(0,xaxis+38.0,511.-yaxis,"%d",point-7);
1410     }
1411     else {
1412         printf(0,xaxis-50.0,511.-yaxis,"%d",point-7);
1413     }
1414     break;
1415 }
1416 return;
1417 }
1418
1419
1420
1421 ockt3(xaxis,yaxis,total,c1,c2,c3,c4,c5,c6,c7,class) {
1422
1423     switch (total) {
1424     case 0:
1425         break;
1426     case 1:
1427         color(c1);
1428         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1429         break;
1430     case 2:
1431         color(c1);
1432         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1433         color(c2);
1434         block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1435         break;
1436     case 3:
1437         color(c1);
1438         block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1439         color(c2);
1440         block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);

```



```

1441     color(c3);
1442     block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1443     break;
1444 case 4:
1445     color(c1);
1446     block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1447     color(c2);
1448     block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1449     color(c3);
1450     block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1451     color(c4);
1452     block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1453     break;
1454 case 5:
1455     color(c1);
1456     block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1457     color(c2);
1458     block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1459     color(c3);
1460     block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1461     color(c4);
1462     block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1463     color(c5);
1464     block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
1465     break;
1466 case 6:
1467     color(c1);
1468     block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1469     color(c2);
1470     block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1471     color(c3);
1472     block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1473     color(c4);
1474     block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1475     color(c5);
1476     block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
1477     color(c6);
1478     block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
1479     break;
1480 case 7:
1481     color(c1);
1482     block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1483     color(c2);
1484     block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1485     color(c3);
1486     block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1487     color(c4);
1488     block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1489     color(c5);
1490     block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
1491     color(c6);
1492     block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
1493     color(c7);
1494     block((xaxis+7)*1.,511.-(yaxis+13)*1.,(xaxis+13)*1.,511.-(yaxis+7)*1.);
1495     break;
1496 default:
1497     overflowtbl[ctroverflow][0]=xaxis;
1498     overflowtbl[ctroverflow][1]=yaxis;
1499     ctroverflow++;
1500     color(c1);

```



```

1501     block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
1502     color(c2);
1503     block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
1504     color(c3);
1505     block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
1506     color(c4);
1507     block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
1508     color(c5);
1509     block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
1510     color(c6);
1511     block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
1512     color(c7);
1513     block((xaxis+7)*1.,511.-(yaxis+13)*1.,(xaxis+13)*1.,511.-(yaxis+7)*1.);
1514     color(class);
1515     afltcolor=class;
1516     if(xaxis>250) {
1517         printf(0,xaxis+38.0,511.-yaxis,"%d",total-7);
1518     }
1519     else {
1520         printf(0,xaxis-50.0,511.-yaxis,"%d",total-7);
1521     }
1522     break;
1523 }
1524 return;
1525 }
1526
1527
1528 reset() {
1529     /* reset function for successive network iterations */
1530     int i,mark,x,y,z;
1531
1532     if(vers==1 || vers==2) {
1533         for(i=0;i<nhrplot;i++) {
1534             color(14);
1535             x=linktbl[i][1];
1536             y=linktbl[i][2];
1537             z=linktbl[i][3];
1538             if(z==0) {
1539                 block((x-3)*1.,511.-(y-7)*1.,(x+3)*1.,511.-(y-13)*1.);
1540                 block((x-13)*1.,511.-(y+13)*1.,(x+13)*1.,511.-(y-3)*1.);
1541             }
1542             else {
1543                 block((x-16)*1.,511.-(y-2)*1.,x*1.,511.-(y-10)*1.);
1544                 block((x-16)*1.,511.-(y+10)*1.,x*1.,511.-(y+2)*1.);
1545             }
1546         }
1547     }
1548     else {
1549         for(i=0;i<tblctr-1;i++) {
1550             color(14);
1551             x=uniquetbl[i][1];
1552             y=uniquetbl[i][2];
1553             z=uniquetbl[i][3];
1554             if(z==0) {
1555                 block((x-3)*1.,511.-(y-7)*1.,(x+3)*1.,511.-(y-13)*1.);
1556                 block((x-13)*1.,511.-(y+13)*1.,(x+13)*1.,511.-(y-3)*1.);
1557             }
1558             else {
1559                 block((x-16)*1.,511.-(y-2)*1.,x*1.,511.-(y-10)*1.);
1560                 block((x-16)*1.,511.-(y+10)*1.,x*1.,511.-(y+2)*1.);

```



```

1561     }
1562     }
1563     }
1564     return;
1565 }
1566
1567
1568 coltab() {
1569     int i;
1570
1571     i=11*16;
1572     lodcol(i++,15,15,15); /* color 0 */
1573     lodcol(i++,0,10,0); /* color 1 */
1574     lodcol(i++,15,0,0); /* color 2 */
1575     lodcol(i++,15,15,0); /* color 3 */
1576     lodcol(i++,12,0,12); /* color 4 */
1577     lodcol(i++,5,5,12); /* color 5 */
1578     lodcol(i++,6,6,5); /* color 6 */
1579     lodcol(i++,5,3,3); /* color 7 */
1580     lodcol(i++,10,6,10); /* color 8 */
1581     lodcol(i++,12,5,5); /* color 9 */
1582     lodcol(i++,5,5,3); /* color 10 */
1583     lodcol(i++,12,12,0); /* color 11 */
1584     lodcol(i++,8,7,3); /* color 12 */
1585     lodcol(i++,5,4,2); /* color 13 */
1586     lodcol(i++,0,0,6); /* color 14 */
1587     lodcol(i++,6,0,0); /* color 15 */
1588     return;
1589 }
1590
1591
1592     /*****
1593     /*****
1594     /*****      END OF PROGRAM LINKGRAPH.C      *****/
1595     /*****
1596     /*****
1597

```


LIST OF REFERENCES

1. Abramson, Norman and Franklin F. Kuo, Computer-Communication Networks, Prentice-Hall, 1973.
2. Anderson, George A, "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples", Computing Surveys, Vol. 7, No. 4, Dec 1975.
3. Awad, Elias M., Systems Analysis and Design, Richard D. Irwin, Inc., 1979.
4. Browne, J. C., "Performance Analysis and Evaluation: The Connection to Reality", Research Directions in Software Technology, The MIT PRESS, 1979.
5. Cox, L. A. Jr., "Performance Prediction of Computer Architectures Operating on Linear Mathematical Models", Ph.D. Thesis, Computer Science Dept., UC Davis Report UCRL-52582 (Sept 28, 1978).
6. Cypser, R. J., Communications Architectures for Distributed Systems, Addison-Wesley Publishing Company, 1978.
7. Davies, D. W., and others, Computer Networks and Their Protocol, John Wiley and Sons, 1979.
8. Dixon, F., "The Impact of Technology on the Future Tactical Air Control Systems", Air Command and Staff College, March 1977.
9. Enslow, Philip E. Jr., "Multiprocessor Organization-A Survey", Computing Surveys, Vol. 9, No. 1, March, 1977.
10. Hamming, R. W., "Why Do You Believe the Simulation is Relevant?", Simulation, Naval Post Graduate School, 1979.
11. Heitreyer, Constance L., and others, "A Survey of Packet Switching Techniques for Broadcast Media", NRL Report 8035, Naval Research Laboratory Washington D.C., Oct 12, 1976.
12. Hopper, Grace, M., "David and Goliath", Selected Computer Articles 78, Dept. of Defense Computer Institute, 1978.

13. Kahn, Robert E., "The Organization of Computer Resources into a Packet Radio Network", IEEE Transactions on Communications, Vol. Com-25, No. 1, Jan 1977.
14. Kleinrock, Leonard, "Performance of Distributed Multi-Access Computer-Communication Systems", Proceedings of IFIP Congress 1977, Toronto, Canada, Aug 1977.
15. Kleinrock, Leonard, "Principles and Lessons in Packet Communications", Proceedings of the IEEE, Vol. 66, No. 11, Nov 1978.
16. Kunzelman, Ronald C., and others, "Packet Radio Experimental Network Research and Development, Final Technical Report", SPI International, Jan 1978.
17. Lawson, Billy R., "A Testbed for Evaluation of Packet Radio in an Army Tactical Corps", U.S. Army letter.
18. Moli, Gesualdo De, "On Networking", Computer Networks and Simulation, North-Holland, 1978.
19. Peterson, James L., "Petri-Nets", Computing Surveys, Vol. 9, No. 3, Sept 1977.
20. Price, W. L., Computer Networks and Simulation, North-Holland Publishing Company, 1978.
21. Stowers, Douglas M., "Computer Architecture Performance Prediction for Naval Fire Control Systems", Naval Post Graduate School Thesis, Dec 1979.
22. Terman, Lewis, M., "The Role of Microelectronics in Data Processing", Scientific American, Sept 1977.
23. Thurber, Kenneth J., and Leon D. Wald, "Associative and Parallel Processors", Computing Surveys, Vol. 7, No. 4, Dec 1975.
24. FM-24-1, Combat Communications, U.S. Army Field Manual.
25. Letter of Agreement (LOA) for PLRS/JTIDS Hybrid, U.S. Army Training and Doctrine Command, ACN 58401, Dept of the Army, 6 July 1979.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	I. Larry Avrunin DTNSPDC Code 18 Bethesda, Maryland 20084	1
2.	Mr. R. P. Crabb Code 9134 Naval Ocean Systems Center San Diego, California 92152	1
3.	Mr. C. H. Gleissner DTNSRDC Code 18 Bethesda, Maryland 20084	1
4.	Ms. Kathryn Heninger Code 7503 Naval Research Laboratory Washington, D. C. 20375	1
5.	Mr. Roanld P. Kasik Code 4451 Naval Underwater Systems Center Newport, Rhode Island 02840	1
6.	Dr. J. McGraw U. C. - L.L.L. (L-794) P.O. Box 808 Livermore, California 94550	1
7.	Mr. Henry G. Stuebing Code 503 Naval Air Development Center Warminster, Pennsylvania 18974	1
8.	Mr. Joel Trimble Code 221 Office of Naval Research 800 North Quincy Arlington, Virginia 22217	1
9.	Mr. Mark Underwood Code P204 NPROC San Diego, California 92152	1

10.	Mr. Michael Wallace DTNSRIC Code 1828 Bethesda, Maryland 20084	1
11.	Mr. Walter P. Warner NSWC Code K70 Dahlgren, Virginia 22448	1
12.	Dr. John Zenor Code 31302 Naval Weapons Center China Lake, California 93555	1
13.	Office of Research Administration Code 012A Naval Postgraduate School Monterey, California 93940	1
14.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
15.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
16.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	2
17.	Prof. L. Cox, Code 52C1 Department of Computer Science Naval Postgraduate School Monterey, California 93940	5
18.	Prof. J. Wozencraft, Code 74 Chairman, C3 Academic Group Naval Postgraduate School Monterey, California 93940	2
19.	Prof. R. Schell, Code 52Sj Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
20.	Prof. O. Kodres, Code 52Kr Department of Computer Science Naval Postgraduate School Monterey, California 93940	1

- | | | |
|-----|---|---|
| 21. | Cpt. Robert J. Hartel
Code 110
Defense Communications Agency
Washington D.C. 20305 | 5 |
| 22. | Capt. Stephen C. Jennings
MCDEC
Communication Officers School
Quantico, Virginia 22134 | 2 |
| 23. | Mr. Douglas M. Stowers
Naval Sea Systems Command
Code Sea 62Y21F
Washington D.C. 20362 | 1 |

Thesis
J4726
c.1

Jennings
Petri-Net simulations
of communications net-
works.

189302

3 NOV 83

JUN 25 85

12 SEP 86

12 MAY 88

25 OCT 88

279101

301137

31543

32941

32612

Thesis
J4726
c.1

Jennings
Petri-Net simulations
of communications net-
works.

189302

thesJ4726

Petri-Net simulations of communications



3 2768 002 10750 0

DUDLEY KNOX LIBRARY